

Evaluating Erlang for Web 2.0 Development

Szymon Piotr Czaja

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

7th of September 2015

Abstract

The robust language architecture devised for distributed environments and massive, near linear, scalability of the Erlang platform make it particularly suitable for the development of modern, rich user experience web applications. Yet Erlang never managed to gain any substantial popularity in web application development. It is seen as a niche language which suffers from a rather unimpressive number of libraries. This work provides a critical insight into the suitability of Erlang for web application development. It is found that the existing attempts of making Erlang a web application development platform have failed to attract wider audience mainly because of efforts of trying to mimic existing, largely outdated industry standards. A new set of standards is proposed that represents a better fit for web development challenges in the face of the most recent technology advancements. The new standards are implemented in an Erlang web application. It is found that the application does not fully meet the requirements formed on the basis of previously formulated standards. The application is benchmarked with a 1000 websocket connection initiation requests per second while keeping established connections alive for 1 minute in order to test its scalability and resilience - key qualities guaranteed by the Erlang OTP. It is found that the application successfully establishes 47520 websocket connections exhibiting high levels of resiliency where a randomly dying process does not have any impact on the running of the system. The development process of the application reveals certain challenges that need to be overcome in order to seriously consider Erlang as a web development platform. These challenges include lack of libraries - particularly database drivers and ORM libraries. An example ORM library is implemented and used in the web application project to provide an abstract interface to PostgreSQL and Mnesia databases. The library demonstrates a way of mapping Mnesia records and PostgreSQL relational data to maps which Erlang started to support in its 17th release.

# Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Signature:

# 

# Acknowledgements

Very special thanks to my wife - **Anna** who supported me in my work on the project,

**Phil Trinder** who guided me through the process of writing this project and

**Keith Langdon** who encouraged me to explore programming in Erlang OTP.

Contents

[Abstract 2](#_Toc429832407)

[Education Use Consent 3](#_Toc429832408)

[Acknowledgements 4](#_Toc429832409)

[1. Introduction 7](#_Toc429832410)

[1.1 Context 7](#_Toc429832411)

[1.2 Problem Statement 8](#_Toc429832412)

[1.2.1 Aims 8](#_Toc429832413)

[1.2.2 Objectives 8](#_Toc429832414)

[1.3 Contributions 9](#_Toc429832415)

[2 Background Survey 10](#_Toc429832416)

[2.1 The Web Is Changing 10](#_Toc429832417)

[2.1.1 Protocol 10](#_Toc429832418)

[2.1.2 Scripting 10](#_Toc429832419)

[2.1.3 Data Serialisation 10](#_Toc429832420)

[2.1.4 Client Side Content Handling 11](#_Toc429832421)

[2.2 Web Application Frameworks Today 11](#_Toc429832422)

[2.2.1 Server-side Web Frameworks 11](#_Toc429832423)

[2.2.2 Client-side Web Frameworks 12](#_Toc429832424)

[2.3 Web Application Development Standards 13](#_Toc429832425)

[2.4 The Erlang Programming Language 13](#_Toc429832426)

[2.4.1 Brief History 13](#_Toc429832427)

[2.4.2 Characteristics 13](#_Toc429832428)

[2.4.3 The Language For The Web 14](#_Toc429832429)

[2.5 Brief Overview Of Erlang Web Framework Projects 15](#_Toc429832430)

[2.5.1 Nitrogen 15](#_Toc429832431)

[2.5.2 Zotonic 16](#_Toc429832432)

[2.5.3 Chicago Boss 16](#_Toc429832433)

[2.5.4 N2O 17](#_Toc429832434)

[2.5.5 Farwest 17](#_Toc429832435)

[2.6 Are Erlang Web Framework Projects Going In The Right Direction? 17](#_Toc429832436)

[2.7 A Brief Reflection On Ruby 19](#_Toc429832437)

[2.8 Limitations Of Erlang 19](#_Toc429832438)

[2.8.1 Erlang Libraries 19](#_Toc429832439)

[2.8.2 NIFs 21](#_Toc429832440)

[2.8.3 BIFs 21](#_Toc429832441)

[2.8.4 String Operations 21](#_Toc429832442)

[2.8.5 Heavy Computations 21](#_Toc429832443)

[2.8.6 Database Drivers 21](#_Toc429832444)

[3 New Web Development Standards 23](#_Toc429832445)

[3.1 Erlang in Web Development 23](#_Toc429832446)

[3.2 Erlang Web Development Standards 23](#_Toc429832447)

[3.2.1 OTP By Default 24](#_Toc429832448)

[3.2.2 Full Duplex Channel 24](#_Toc429832449)

[3.2.3 Solid Foundations 24](#_Toc429832450)

[3.2.4 Pluggability 24](#_Toc429832451)

[3.2.5 Erlang and JavaScript 24](#_Toc429832452)

[3.2.6 MVC 25](#_Toc429832453)

[3.2.7 Audience 25](#_Toc429832454)

[4 Erlang Web Application Requirements 26](#_Toc429832455)

[4.1 Non-Functional Requirements 26](#_Toc429832456)

[4.2 Functional Requirements 26](#_Toc429832457)

[5 Implementation 27](#_Toc429832458)

[5.1 High Level Architecture 27](#_Toc429832459)

[5.2 Client Application 27](#_Toc429832460)

[5.3 Application Server 28](#_Toc429832461)

[5.3.1 Soil 28](#_Toc429832462)

[5.4 Database 29](#_Toc429832463)

[5.4.1 Norm 29](#_Toc429832464)

[6 Evaluation 31](#_Toc429832465)

[6.1 Requirement Status 31](#_Toc429832466)

[6.1.1 Status of Non-Functional Requirements 31](#_Toc429832467)

[6.1.2 Status of Functional Requirements 36](#_Toc429832468)

[6.2 Recommendations 36](#_Toc429832469)

[6.2.1 Libraries 36](#_Toc429832470)

[6.2.2 Erlang ORM 37](#_Toc429832471)

[6.2.3 Code Repository 38](#_Toc429832472)

[6.2.4 Release Tool 38](#_Toc429832473)

[6.2.5 Testing 39](#_Toc429832474)

[7 Conclusion 40](#_Toc429832475)

[7.1 Summary 40](#_Toc429832476)

[7.2 Limitations 40](#_Toc429832477)

[7.3 Further Work 41](#_Toc429832478)

[7.3.1 Further Work On This Project 41](#_Toc429832479)

[7.3.2 Further Improvements To Erlang 41](#_Toc429832480)

[7.4 Challenges For The Erlang community 42](#_Toc429832481)

[Appendix A - List Of Figures 43](#_Toc429832482)

[Bibliography 48](#_Toc429832483)

# 1. Introduction

## 1.1 Context

Some might argue the HTTP 1.1 is the standard and it is too early to talk about a new web communications protocol in the context of web application development. Yet the specification for the new protocol running the World Wide Web was recently finalised [1] and in the coming years the number of websites running on HTTP/2 will only increase. The introduction of the new protocol impacts on the way web applications are developed as HTTP/2 enforces a single, persistent, bi-directional communication channel between the server and a client, which represents a major advancement over the currently adopted request-response paradigm. The emerging requirement of web application servers to have the ability of easily maintaining and communicating between potentially very large numbers of concurrent bi-directional connections, brings web application development to a whole new level. This shift in client-server communications is a great opportunity for concurrency-oriented, functional languages to become the default choice programming languages for web application projects. Erlang being built for concurrent, real time communications is an example of such a language. The following chapters evaluate Erlang in the context of web application development. A comprehensive survey is provided on the current state of web technologies, the standards adopted in the web development industry, the most popular web application frameworks and the work carried out in the Erlang community that aimed to popularise the practise of developing web applications in Erlang. A set of new web development standards is formed which aims to represent the challenges that modern web applications will have to meet in the imminent future. These standards are later applied to a concrete implementation of an Erlang web application in an attempt to empirically assess the suitability of OTP platform for web development. The experiences and observations gained as part of the development process provide grounds for a set of recommendations aiming to make Erlang OTP a more attractive platform for web developers.

## 1.2 Problem Statement

### 1.2.1 Aims

This software development project has three aims:

1. To form a set of web development standards that better reflect the trends in the industry and account for the recent technology advancements.
2. To design and implement a concrete solution that follows the new standards in Erlang.
3. To evaluate the development process and form a set of recommendations which would help to elevate Erlang OTP as a web development framework.

### 1.2.2 Objectives

In order to meet the first aim of this paper an extensive background survey will be carried out. The survey will provide a general insight into the current state of web application landscape with a particular focus on the technologies that govern the delivery of content around the web. This will be followed by a brief overview of the most popular web development frameworks today along with a short summary of their main features. This will help to identify common characteristics between them and allow for a high level summary of web development practises as promoted by the most popular web development tools. Following a brief account on the current state of web development standards, the survey will turn to the realm of Erlang programming. The survey will outline main characteristics of Erlang emphasising capabilities as a platform for web application development. This will be followed by an outline of the most prominent Erlang web frameworks projects, along with a short assessment of the impact they have made in a wider web application development community. Having introduced Erlang and the most popular Erlang web framework projects, a short insight will be made into Ruby on Rails - a web development platform that is based on a language in many ways similar to Erlang. The background survey will conclude with a brief summary of the main limitations of Erlang language and will effectively provide foundations for the development of a new set of web development practises. This new set of standards will be derived from the key facts discovered in the course of the background survey, mainly as a result of an evaluation of the existing web application development landscape, with a special insight into the efforts made by the Erlang community. The second aim involves the development of a modern Erlang web application that has been implemented according to the new development standards formed as a result of evaluation described above. A set of requirements for a the Erlang application will be formulated in order to realise this aim. Once the development process of the website is completed the project will be evaluated in the context of the status of requirements which have and have not been met. A reflection upon the implementation process along with the status of the requirements will allow the formation of recommendations for web application development in Erlang, which will effectively realise the third aim of this paper.

## 1.3 Contributions

This section lists the key contributions of this project.

1. Reflection upon the current state of web technologies - with a particular focus on the new HTTP/2.0 protocol, the rising popularity of client side JavaScript web frameworks, the new revision of JavaScript language and greater reliance on binary rather than text based data serialisation formats.
2. Description of Spring, Django, Symfony, Ruby on Rails and Play as examples of the most popular web application frameworks today, for the purpose of identifying common features across them and crystallising current standards adopted in the industry.
3. Introduction to Erlang detailing characteristics of the language that are most relevant in the context of web development along with the language's limitations and a survey of Erlang web framework projects i.e. Zotonic, Chicago Boss, Nitrogen and N20 in order to provide a commentary on some key design decisions as realised in the discussed frameworks.
4. Proposal of new web development standards better suited for the web development challenges of the future web applications along with a concrete implementation of these standards in a modern web application built in Erlang, which is benchmarked with 1000 websocket opening requests per second in order to test its scalability and resiliency - one of the key qualities guaranteed by the Erlang Platform. An ORM library is also developed to provide an abstract interface to a SQL database utilising a novel approach of representing relational data in Erlang using maps which were recently added as part of EEP43 [2].
5. List of recommendations for Erlang platform to make it a more appealing web application development framework which address the lack of libraries, the functional nature of the language, the lack of Erlang code repository, the release handling and the testing. Recommendations are accompanied by a list of suggestions as to the further work needed for the Erlang system, as well as within the community.

# 2 Background Survey

## 2.1 The Web Is Changing

Technology giants such as Microsoft, Google and Apple run campaigns promoting their mobile devices and internet browsers in an attempt to dominate the internet and dictate the rules governing the exchange of information on the web. The web today is already mobile-first. Big and small organisations develop their software with view for use on mobile devices first. Web sites need to easily adapt to screens of varying sizes, have small data footprint and response times. In 2008 Amazon's research showed that every 100 ms increase in load time results in a decrease in sales by 1% [3]. In order to achieve effectively a faster web the current web technologies needed to evolve. The underlying protocols need to run with improved performance, the client side scripting needs to mature into a well structured and reliable coding language, the encoding of the data needs to be more efficient and the web development in general needs to adapt to the changing landscape. The web today is in a process of transition and the change is happening on all fronts.

### 2.1.1 Protocol

Not long ago AJAX was the king, today depending on the requirements AJAX, long polling, server-sent events and Websockets are used with conjunction with each other to deliver content to browsers and web application clients running on a variety of devices. In the meantime Google has been working on SPDY protocol [4] which lay the foundation for the new HTTP standard that supersedes the HTTP/1.1 version currently in use. HTTP/2 specification has recently been finalized [1] and the following RFC document is soon to be released. The HTTP/2 represents a fundamental change in the web communications. The new standard means moving away from the old stateless paradigm in favour of stateful, full duplex communications. For the end users this means a much better experience - mainly faster and more responsive browsing. For the developers it is entirely new way of thinking about web development.

### 2.1.2 Scripting

The invention of AJAX and NodeJS [5] substantially fuelled JavaScript's popularity. From a language that most programmers looked down at JavaScript became a serious coding platform that big companies started to use for serious projects. More and more websites use client side JavaScript libraries that manage content delivery at runtime. These libraries have grown into powerful client side web development frameworks, capable of processing and rendering data as well, as it was once possible only on the server side. It quickly transpired that the language which had not changed much since 1999 needed a new specification in order to stand up to the raising expectations. The final specification of the new standard was published in June 2015 [6]. JavaScript has undergone a major change, in many respects truly resembling the semantics of Java and bringing in a range of improvements including better support for internationalisation and code isolation. All major client side web frameworks such as AngularJS[7] or Ember.js[8] are in the process of adopting of the new standard.

### 2.1.3 Data Serialisation

The limitations of XML and JSON are very apparent. Even though the standards do not seem to be losing their popularity, more and more companies tend to look for more elegant and lightweight serialization formats. While JSON might be a convenient format to work with in the browser it is largely inefficient when serialized into a string and exchanged over wire. The same holds true for XML. An average website today is much more interesting that an average website five years ago and the expense associated with the serving of the right data in the right time plays a crucial role. More and more companies find themselves looking for ways to optimise the exchange of data and make it more supported between different platforms thus new serialization formats e.g. BSON [9], Thrift [10], Protocol Buffers [11] and new RPC protocols such as BERT-RPC [12].

### 2.1.4 Client Side Content Handling

The web started with the idea of serving static HTML files onto the browsers. Over time it became clear what is needed is dynamically generated content. This was first accomplished by the way of running server side scripts that dynamically generated a whole new page and served it onto the client. Later - with the invention of AJAX - page rendering became more sophisticated with the browser requesting smaller parts of the page and reloading only those DOM elements that were changing. The websites today are a mixture of static and dynamic content with parts of the page fetched on demand when needed, and reloaded almost real-time. Since more and more websites run powerful web development framework libraries on the client side the data processing and rendering is handled mostly in the browser. There is no need any more to generate HTML on the server. The server it left with its true purpose - to serve data. All data transformation, formatting and rendering is done on the client side. Although HTML rendering libraries like Django [13] or JavaServer Pages [14] will continue to be used their popularity will diminish.

## 2.2 Web Application Frameworks Today

The process of developing a web application quickly reveals that there is a lot of 'boiler plate' code. The programmer finds him or herself repeating the same tasks e.g. processing a HTTP request, accessing a database or updating an HTML template. These tasks could be separated into their own domains such as the routing of the request, the database operations and the rendering of the HTML page. Web application frameworks deliver the separation of concerns in the development process and the code base for most common operations. Employing a rich and flexible web framework reduces the time needed to develop a website and minimises the chances of running a bad code. The frameworks strive to evolve in line with the requirements of modern websites, although the approach to the aiding of the developer in the meeting of those requirements varies across the frameworks.

### 2.2.1 Server-side Web Frameworks

#### 2.2.1.1 Spring

The Spring framework [15] is written in Java which is a cross platform imperative language, designed by James Gosling and first released in 1995. Java is mainly object oriented and the latest release added a lot of support for concurrency [16]. The Spring is very rich with many tools available for web services, data access, authentication, authorisation and messaging. Most of the SQL and noSQL database vendors ship drivers for the Java platform. There are also quite a few templating engines such as JavaServer Pages [14], Velocity [17] or Freemaker [18] and front end libraries e.g. RichFaces [19] or OpenFaces [20]. Spring is one of the most popular web development frameworks that is widely adopted particularly in enterprise environments. The Spring libraries that the framework is built upon are used in popular Enterprise Service Buses (ESBs) such as Mule [21] or JBoss [22].

#### 2.2.1.2 Django

Django [23] relies on an imperative, dynamic and multi-paradigm (object oriented, functional and procedural features) language first engineered by Guido van Rossum in 1991. Django is web server agnostic though may require some additional Python libraries for running. The framework officially supports only relational databases and these are PostgreSQL, MySQL, SQLite, and Oracle [24]. There are however some non-official drivers for noSQL data stores available from the Python repositories. Python is known for a wide range of packages aka libraries available for the platform thus some boiler plate code can be easily obtained for implementing web services or adding websockets support. The most well known Django customers include Pinterest [25] and Instagram [26].

#### 2.2.1.3 Symfony

The framework [27] is based on PHP which was created by Rasmus Lerdorf in 1995. PHP originally started as a scripting language and only eventually became general-purpose. It is an imperative language which exhibits procedural, functional and object oriented paradigms. The Symfony uses Doctrine to map PHP objects to relational databases. It supports MySQL, PostgreSQL or Microsoft SQL. The Doctrine ODM library provides access to a noSQL data store MongoDB [28]. The Symfony2 uses a template rendering library named Twig which mimics the one used in Django projects. PHP can be embedded in the HTML code which can in turn run JavaScript what gives the programmers a lot of flexibility in wiring the front end to the back end web application. Symfony is used by Delicious [29], Yahoo! Bookmarks [30] and OpenSky [31].

#### 2.2.1.4 Ruby on Rails

RoR [32] runs on Ruby which similarly to Python is an imperative, dynamic, multi-paradigm programming language designed by Yukihiro Matsumoto in the middle of 1990s. RoR is web server agnostic and is typically deployed with a relational database although the majority of database vendors provides Ruby drivers for their products. RoR delegates the handling of user interaction to JQuery. Initially advocating SOAP to support web services, it later gave more focus to REST. Despite some issues with the Ruby interpreter related to scalability which caused Twitter to partly move to Scala [33], RoR is widely used by such giants as GitHub [34], Scribd and Groupon [35].

#### 2.2.1.5 Play

The framework [36] was written in Scala [37] which is an object-functional programming language with support for concurrency that runs on Java Virtual Machine. The language was designed by Martin Odersky and had its first stable release in October 2013. Scala is best known for its implementation of the actor model [38] which in many ways mimics the design choices made in Erlang OTP. Since Scala runs on the JVM the Play framework benefits from the large number of Java libraries widely used by the Spring framework. The framework and Scala language run Twitter [33] and Guardian [39] websites.

### 2.2.2 Client-side Web Frameworks

#### 2.2.2.1 AngularJs

Written in JavaScript which initially started as a scripting language and eventually became a multi-paradigm, general purpose programming language. JavaScript was developed by Brendan Eich and first appeared in 1995. AngularJs [40] is different from other frameworks discussed. It is a client-side web framework, meaning it is meant to be run in the browser. AngularJs therefore does not maintain connections to databases or expose webservices. It is fed with the data from the server, creates models, and renders views by manipulating the DOM within the browser. It is a Google project which recently became very popular mainly due to the balance between the ease of use and rich functionalities offered.

## 2.3 Web Application Development Standards

The above short and high-level overview of the most popular web frameworks revels certain key features that all of them share. All aforementioned frameworks are stable and widely supported. They all employ the Model-View-Controller (MVC) methodology for implementing user interfaces. The view handling logic is clearly separated from template rendering. Most of them integrate well with noSQL databases with either built in drivers or external libraries, all work with relational databases as standard. None of the frameworks relies on one specific web server and all have a rich collection of extra packages and libraries to choose from. One of the main reasons behind the success of Scala is the usability of existing Java libraries. All support both XML and JSON for the data exchange format between the client and the server. Finally, they all attracted a high profile customer.

## 2.4 The Erlang Programming Language

### 2.4.1 Brief History

The Erlang programming language has been engineered in Ericsson laboratories by Joe Armstrong, Robert Virding and Mike Williams for the purpose of programming telephone switches [41]. The project was launched in 1994 and was quickly augmented with the Open Telecoms Platform (OTP) in 1996. The OTP is a design methodology and a set of libraries for building robust systems that make Erlang a platform rather than merely a programming language. The platform was released as an open source project in 1998. Since the initial release the project has been actively maintained and supported.

### 2.4.2 Characteristics

The peculiar set of characteristics of the language is the main factor contributing to the relatively low memory consumption and exceptional performance when running on multiple processors.

#### Functional

Erlang does not have mutable variables and it avoids state. The functions in Erlang are first-class i.e. they are treated like values thus can be passed as arguments, included in a list or returned from other functions. The programming in Erlang is achieved with expressions. The actual internals of how the desired results are delivered are hidden from the programmer. Erlang ships with a number of built-in-functions (BIFs) that provide e.g. typecasting, process spawning and monitoring.

#### Dynamic

The run-time computations may throw type exceptions. There is no mandatory type checking at compile time. The dialyzer tool [42] is made available to Erlang developers to perform the analysis of the code in respect of any type cast exceptions.

#### 

#### Strongly typed

The type system in Erlang cannot be subverted. Dissimilar types require explicit type casting what is achieved using built-in functions (BIFs).

#### Concurrent

Erlang provided support for concurrent processing from the very beginning. It also quickly started to support the multi-core, parallel computations. Due to orienting of the language for concurrency from the very beginning, the adoption of parallel processing in Erlang virtual machine made no difference in the way of coding for the programmers. Eckel et al explicitly recommends writing highly concurrent applications in Erlang rather than in Java [43]. The concurrency has been implemented in Erlang using the actor model initially proposed by Hewitt [38], thus the communication between Erlang processes is done via an asynchronous message passing though synchronous communication is very well supported.

#### Interactive

Erlang ships with an interactive shell the user can attach to and interact with the platform.

**Server side orientated**

The language has been designed for running on middleware and backend hardware. There are few libraries provided for coding Graphical User Interfaces (GUIs) in Erlang.

### 2.4.3 The Language For The Web

Since the open source release of the Erlang platform, it quickly attracted a lot of interest due to its exceptional features such as fault tolerance, hot-code swapping and numerous optimizations for concurrent and distributed environments. These features make Erlang a perfect fit for web application development.

#### Fault tolerant

Erlang programs written according to the OTP methodology structure processes in trees of supervision, giving each process a role of a worker or a supervisor. The supervisors implement strategies for dealing with crashes within their children. When one is detected the supervisors can restart the crashed processes or bring the whole tree down and restart it again. Being fault tolerant is one of the basic requirements when programming for the web where delays, timeouts, unresponsive services or node failures are usual occurrences.

#### Scalable

The processes in Erlang do not share memory, the communication is done via message passing and there is no implicit synchronisation. The spawning of processes is instant and asynchronous unless synchronisation is needed. Scalability is the main issue for web applications which may grow rapidly in popularity. Since the number of users of a website may almost instantaneously grow into a number the website has never seen before, scalability on the web is a key. The current practice in the industry is to spend more money on the hardware in line with growing demand for the service. Having scalability in mind early when the prototype of the application is built may bring substantial savings at later stages when the project is live in production environment.

#### Highly available

Along with the OTP process supervision, the platform provides hot-code swapping which allows administrators to swap live Erlang code while the application is running. This means dynamic upgrades and live patching with zero down-time. This is another feature of the Erlang language that makes it very well suited for web development especially in situations when there is a Service Level Agreement in place specifying the web service provision guarantees.

#### Massively concurrent

The processes are cheap and easily managed by the VM also on multi-core processors therefore allowing the platform to handle millions of simultaneous requests. One of the main strengths of Erlang comes from the fast spawning of new processes and ease of communication between them. Although some might argue that the message passing that Erlang uses to communicate between the process is expensive, the support for efficient exchange of messages between many web application clients each running often more than one process on its own is extremely valued in the time of great popularity of social media applications.

#### Soft real-time

The web is soft real-time. The requests for resources on the web have execution time associated with them and the users expect that. Taking a second more or less to deliver the resource does not crash the web service or prevent the users from using it. Erlang OTP built-in process scheduler is a good fit for soft real-time communications, switching between process and ensuring no single process locks on more resources than others.

## 2.5 Brief Overview Of Erlang Web Framework Projects

In early days of web development with Erlang the LYME acronym was coined which stands for Linux, Yaws, Mnesia and Erlang and which was supposed to describe a stack competing with the LAMP standard - Linux, Apache, MySql and PHP - the most up-to-date web framework projects tend to develop their own stacks that often allow for a greater choice in the pluggable components. In fact the Yaws web server which was a cornerstone of LYME and still remains considered the most matured Erlang web server, lost its popularity and is often offered as an option rather than the default. For convenience and for the purposes of this paper the term web framework shall be understood as suite of applications that aim to deliver a complete web application running on a stack conceptually similar to LYME or LAMP. There are few popular, actively supported, open source projects which aim to morph into mature Erlang web frameworks. These projects amalgamate the existing libraries and smaller projects and work towards improving them and delivering a coherent and reliable product.

### 2.5.1 Nitrogen

Started in 2008, the framework offers the programmer a choice of few webservers. It supports Inets [44], Cowboy [45], Yaws [46], Mochiweb [47] and Webmachine [48]. Inets is the standard Erlang built-in HTTP server that is shipped with the platform. Cowboy became popular due to its lightweight implementation and great performance under load. A number of Erlang web server benchmarks [49] shows that Cowboy is one of the fastest Erlang HTTP servers although Vionski who had been an active contributor to the Yaws project, pointed out the focus should be on improving the underlying Erlang HTTP libraries rather than measuring how one Erlang web server is better than another [50]. Yaws is the oldest Erlang HTTP server project. It is mature and has a lot of built in functionalities e.g. it supports SOAP, these however make it relatively heavy and resource hungry. Mochiweb is described as "an Erlang library for building lightweight HTTP servers" [51] and Webmachine is built on top of it to provide a layer that adds HTTP semantic awareness. Both libraries have been developed at Basho - a Cambridge, Massachusetts based company, the creator of a distributed, highly available Riak database. The framework is event-driven and uses AJAX and long polling (Comet) to facilitate a text-based communication with the client. The browser generates events which are pattern-matched in Erlang modules on the server, processed and pushed back to the client as JavaScript snippets executed by the browser upon arrival. Nitrogen uses its own Domain Specific Language (DSL) for templating based on mapping DOM elements to Erlang records. This is a convenient way to quickly add new, dynamic HTML elements on the page and process any events attached to them. The framework has not been designed with the support of any database in mind. It does not provide any support for working with models. The plugging of a database backend works the same way as for any other Erlang project.

### 2.5.2 Zotonic

The framework is tightly coupled with the Webmachine HTTP server. Initially, the Webmachine did not meet some of the framework's requirements so the Zotonic team re-factored the code, introduced some optimizations that made the server run faster and released their work as a separate GitHub repository giving it a name Webzmachine. Zotonic was in many ways inspired by Nitrogen. It is event driven and similarly to Nitrogen pushes JavaScript to clients. Zotonic however takes the event-driven communication with the browser a step further. It uses Websockets by default providing a fall-back mechanism to Comet and XML HTTP Request (XHR) automatically by default. The Zotonic's client side JavaScript is also much richer than the client in Nitrogen allowing client side programmers to hook into the Zotonic's event manager. The framework uses an excellent, actively supported ErlyDTL templating library [52] which puts the creation of client side HTML templates in line with the templating system used in Django. Zotonic has been envisioned not only as a web framework but also a Content Management System (CMS). Therefore, it provides facilities to quickly update views, add and remove models and include extra components on the page. The Zotonic makers claim that a simple dynamic website can be created without a single line of Erlang code. While this is an added value to the product, it forces the framework to be heavily reliant on one particular data store. Zotonic supports only PostgreSQL and is tightly coupled with that database. Some features that make the framework stand out are modularity which enables developing custom Zotonic packages that extend the framework with extra functionalities and the ability to run more than one website on a single machine. The CMS features, modularity and the ability to run a number of websites make Zotonic more of a complete system rather than just a set of tools.

### 2.5.3 Chicago Boss

The project quickly became quite popular since its initial release in 2010. The Chicago Boss uses the Simple Bridge library [53] (also used by Nitrogen) to provide a common interface for communicating with multiple Erlang HTTP web servers. The Chicago Boss is also event driven, it supports Comet and Websockets, although unlike Zotonic, there is no default fallback to inferior technology what is largely due to the lack of any specific Chicago Boss JavaScript library on the client side. This on the other hand promotes more flexibility in web application design and naturally pushes the website designer to choose a tested and widely accepted solution such as the jQuery library [54]. The disintegration with any specific client communication handling mechanism allows for more flexibility in the format of messages exchanged between the browser and the server. Similarly to Zotonic, the framework uses the ErlyDTL library to provide Django-like template rendering. Unlike the CMS-enriched system however, Chicago Boss supports a pluggable both noSQL and SQL databases via BossDB library [55] which is actively supported by inter alia the author of the O'Reilly handbook "Building Web applications in Erlang" [56] Zachary Kessin. There is an ongoing debate as to how much you can really abstract from as the SQL and noSQL databases are fundamentally different, yet it is really heart-warming to see efforts being invested into this area as it gives hope of clearly marking the limits of an abstracted data store interface.

### 2.5.4 N2O

This is the most recent Erlang web application framework project started in 2013. N2O has been inspired by Nitrogen. The framework uses a templating system and the Nitrogen DSL to provide an event-driven interaction between the browser and the server by the way of embedding active DOM components in HTML templates via the Nitrogen Erlang records. Since the main aim of the framework however is speed optimisation, the communication with the browser is taken a step further. N2O uses Bullet library [57] which provides the fall-back mechanism featured by the Zotonic framework, from websockets connection to Comet and XHR. The speed improvement is achieved via optimisations in underlying communication format. N2O is the only framework that exchanges Binary Erlang Terms (BERT) encoded messages, a technology originally developed at GitHub to improve the load times of their pages [58]. This is also the only framework supporting dynamic Erlang to JavaScript compilation via a Shen library, a functionality addressed by a number of GitHub projects e.g. LuvvieScript [59]. In an attempt to achieve the maximum speed the authors heavily integrated the project with the Cowboy web server and key-value rather than SQL or document based data stores. N2O consists of Objective-C NSKeyValue coding protocol inspired KVS library [60] that aims to standardise the way of working with key-value based data structures. The library can be regarded as a specialised, key-value equivalent of the BossDB. In general the framework promises to deliver an exceptional speed performance at the expanse of strong coupling with BERT enhanced, Bullet and jQuery based, client side JavaScript libraries and key-value databases.

### 2.5.5 Farwest

Farwest project [61] never took off i.e. it ended only with an idea and never really attracted a community that would work together to produce some code. The project is nonetheless worth mentioning as it managed to gather much attention for a number of reasons. Firstly, because it was conceived by the creator of the most popular Erlang HTTP web server today. Loïc Hoguin is a very devoted and active member of the open source Erlang community who has initiated a number of successful Erlang projects. Secondly, because Loïc managed to run a successful crowdsourcing campaign and collected almost 5000 euros from donations made by various member of the Erlang community [62]. The author however put the project on hold for no obvious reason and is not officially doing any Farwest development anymore. Loïc advocated the idea of clearly dividing the back end development in Erlang from the front end development in JavaScript. He wanted to build a framework that would not require front end developer to have any knowledge of Erlang and simply focus on the client side. Farwest was also envisioned - similarly to Zotonic - to have a system of plugins. The client side was to communicate with the server over a well defined RESTful API [61].

## 2.6 Are Erlang Web Framework Projects Going In The Right Direction?

Despite several attempts to build a popular Erlang web framework, and even though there are already few fairly mature web framework projects out there, all projects appear to have terribly failed. Some might be tempted to explain the failure using the 'Erlang is a niche language' argument, yet this argument should be firmly rejected. Every major coding language today was a niche language at some point and some of them evolved to become industry leaders and some have died or are at the verge of dying. Erlang has shown an incredible resistance to changing trends and is certainly more likely to attract more attention rather than to fade away.

All discussed web framework projects managed to gather a vibrant community of Erlang developers, this however did not translate into creating a product that would become a default choice for building web applications in a wider Erlang community not to mention reaching to wider audience. This is not to say that one has to win the Erlang enthusiasts first before having chances elsewhere. The Erlang community has its own characteristics that make it very distinct from other developer groups. An average Erlanger is usually 'a good all-rounder' as the creator of the Zotonic framework Marc Worrell put it in one of his emails to Zotonic mailing list. Erlangers are open-source enthusiasts who are never comfortable with a technology lock-in and therefore always trying new things and experimenting with new software. This is a tough community to win over and Erlang managed to do just that. One might claim there are developer communities that are easier to impress - yet none of the Erlang web framework projects managed to do it.

Zotonic has had the most intense promotional campaign amongst the discussed frameworks and runs probably the largest number of live websites. When talking about big production deployments the authors refer to a case where Zotonic was used to power a web site that run a large voting campaign [63]. This however was a onetime endeavour. The other websites that the authors put forward are rather unimpressive, not attracting high traffic and often static content oriented. The creators of Zotonic naturally endeavoured to attract the widest possible audience and therefore decided to make the platform an interesting choice for novice web developers, experienced Erlangers as well as people with no programming experience whatsoever. As such, they created a product that was closer to a Content Management System rather than just a web development framework and decided to market is as such. These might attract some potential users however keeping Zotonic more CMS orientated makes it less flexible for advanced Erlang users who want to build their applications taking the full advantage of the Erlang platform avoiding any architecture and design constraints. On the other hand people looking for black boxes that would run their websites are more likely to use the most popular and well supported products such as Drupal [64], Joomla [65] or Wordpress [66] rather than explore such exotic grounds like Erlang. Zotonic proves the approach of trying to appeal to a varied audience is not a valid one.

Chicago Boss has targeted a much more selected audience from start - being a pure web development framework the author of the project aimed to attract a wider community of Erlang enthusiasts and he has certainly succeed at doing that. The project is maintained and extended by a vibrant community made up largely of Erlang enthusiasts. While this keeps the project going it does not necessarily ensure best quality of the code. The author of the project adopted a very pragmatic approach from the start i.e. it is better to deliver something that works on time, rather than to get stuck in the search for the most elegant solution. As much as a valid approach this may have been it did not help the project in the long run. Many experienced developers who became involved in the project quickly started to discover that there are better ways of doing things and instead of fixing what was already out there it was easier to move to another project. Unfortunately, among the people abandoning the project was also its author [67].

Nitrogen is developed by a smaller community than Zotonic or Chicago Boss. The targeted audience are rather advanced Erlang programmers. The project is continually maintained and is the only one amongst the discussed that passed the 1.0 version mark. The creators of Nitrogen invested a lot of effort into the translation of Erlang records into JavaScript code. This allowed for an event driven communication between the server and the browser and almost no need to write any client side code. It is difficult to find much evidence of Nitrogen running any large website. There is not much evidence the idea of writing JavaScript code in Erlang gained any substantial popularity either.

## 2.7 A Brief Reflection On Ruby

Ruby was a great success. The language might be fading away now and its future is unknown [68]. Nonetheless it was very inspiring to see a language with such humble origins as Ruby to achieve such popularity in web development. Perhaps unsurprisingly one might notice Ruby and Erlang have a lot in common. Both languages are open source, dynamic, garbage collected and both run their own VMs. In fact they even conceived a child. The robustness of Erlang and the elegance of Ruby gave birth to Elixir which aims to bring the best of the two languages together [69]. Ruby is a great case study for Erlang as an example of a language which managed to shine in web development. Ruby started as yet another programming language project and its success is mainly attributed to the development of Ruby on Rails or simply the Rails framework ten years later [70]. The creator of Rails David Hansson chose Ruby for a number of reasons though it was mainly the declarative nature of the language that meant accomplishing a lot with only one line of code [71]. What Hansson had at hand when he sat down to the development of Rails and what Erlang programmers are currently missing are reliable and well-tested libraries. The early Ruby community building the Ruby empire was not much different from the current Erlang community. Both were made up by open minded technology enthusiasts, often young with some spare time at their hands willing to explore new territories and experiment with something fresh. This begs the question why there is not that many Erlang libraries out there? The availability of a variety of robust libraries is what sets Ruby apart from Erlang. In order to built a successful web framework there are more than the variety of building blocks needed. Stitching a web application together is about abstracting and making decisions. Rails owes its success to certain principles that the author decided to incorporate into the framework. The two most significant are the 'coding by convention' and 'do not repeat yourself'. These effectively require the users to write their code into a certain manner, disallowing them or making it harder to deviate. However harsh this may sound liberty is a double edged sword and claiming a part of it from anyone choosing Rails was the right thing to do which turned out to contribute to the overall success of Ruby.

## 2.8 Limitations Of Erlang

The lack of any substantial popularity of Erlang based web framework projects may be partly explained by the language limitations, this section presents some of the most apparent.

### 2.8.1 Erlang Libraries

#### Quality

The declarative nature of the language means that some of the lower level function implementations are hidden from the programmer. In Erlang, most of the lower level code is contained in the base platform libraries. These libraries need to be actively maintained at the highest standard in order to achieve the cross conformance between the Erlang platform objectives and the compliance with the most recent communication standards and protocol specifications what usually boils down to keeping up to date with the most recent revisions of RFC papers. The failure to keep the base libraries up-to-date may have substantial consequences. For instance some versions of the Crypto library failed to implement some cryptographic algorithms used by most of the browsers for SSL handshakes [72]. The failure to support the required cipher algorithms in the 16B01 and 16B02 releases of Erlang directly affected communications with iPhone smart-phones [73]. The problems with the implementation of SSL/TLS protocols appear to be an ongoing issue for the Erlang platform. The authors of a popular XMPP server Ejabberd written in Erlang decided to ship their product with a homegrown SSL library rather than using the Crypto library which is delivered as standard with the platform. The consequences of relying on the base Erlang OTP cryptographic libraries should not be presumed negative by default. In fact, recently discovered heartbleed bug in the OpenSSL library used by most of the websites including Amazon and Google does not affect communications built on top of Erlang OTP platform as the SSL libraries used in Erlang did not suffer from the vulnerability [74].

#### Lack thereof

The lack of libraries is voiced around by those who became involved into programming in Erlang [75]. This is surprising considering how long has Erlang been around and deserves a careful consideration. Developers write libraries for different reasons. These could be a desire to learn and develop or more often having a genuine need for a library that performs a specific task or somehow enhances or aids software quality. Erlangers find themselves writing libraries from scratch either because they have not found what they have been looking for or because they have used an existing library only to realise its limitations and therefore deciding to start a new project on their own. Considering how long has Erlang been around it would not be unreasonable to expect that such process would eventually result in a fairly large base of stable and well tested software that the rest of the community could use. Indeed Erlangers enjoy some really great libraries they can download for free. The problem is not in the lack of libraries per se as in the majority of cases developers find evidence of someone somewhere trying to solve the same or a similar issue and therefore writing some code. The problem is in the lack of usable libraries.

#### Writing Usable Libraries

Writing usable libraries in Erlang is problematic because Erlang is more than a language - it is also a platform and in Erlang world the difference between an application and a library is very sublime and unfortunately this is not emphasised enough among the members of the community. In fact the terms dependency, library and application are often used interchangeably. This might be acceptable among Erlang experts who have a thorough understanding of OTP, experience writing Erlang code and running ERTS [76] in production. However for open source enthusiasts who often learn by example from code snippets and random articles around the web it is crucial they understand they are potentially writing a library that may be used in a large system that does a lot of parallel computations which needs to scale. Erlang is a language and a platform for writing highly concurrent and scalable software - this does not mean that these qualities are implicitly incorporated into every Erlang program. Working on a potentially large system requires a series of good design decisions and writing a usable Erlang library requires a good understanding of the potential use cases. It is no surprise that the most popular Erlang libraries are blocks of procedural code executed as part of an OTP tree of the host application. Writing a usable application that spawns its own OTP tree that can be safely embedded into a larger system requires advanced skill and knowledge in order to avoid bottlenecks, mysterious crashes and difficulties debugging the code. There is extra complexity involved if the application is required to run in a distributed manner. Finally it is important to realise there are things Erlang is not good at and therefore it may be more effective to write some routines in a language more suitable for the given task.

### 2.8.2 NIFs

Erlang supports Natively Implemented Functions which represent a great way of extending Erlang with imperative, C-implemented code. While NIFs are an interesting way to speed up Erlang computations, their use is generally considered harmful. Firstly, executing a NIF is done at the OS rather than Erlang VM level. Managing multiple OS processes by the system scheduler is costly and bringing about switching between Erlang VM and the system scheduler essentially removes Erlang's claims of scalability. Secondly, a crashing NIF results in bringing the whole Erlang VM down therefore precluding the high-availability of the system.

### 2.8.3 BIFs

The underlying C implementations of Erlang kernel Built-In Functions suffer from performance optimisations. The makers of Cowboy web server note that avoiding the use of one of the BIFs instantly brought a great performance boost [77].

### 2.8.4 String Operations

Strings or array data types do not exist in Erlang. Characters are integers and Strings are lists of integers. This means that operations on strings are operations on lists, which for long lists can be quite slow and resource consuming. It is generally accepted that Erlang is not good for string manipulations. This has quite serious implications for web development, mainly because the most popular web data formats such as JSON or XML are text based. Although the handling of JSON and XML can be easily achieved with the use of some extra libraries such as Mochijson2 [78] or Erlsom [79] respectively, the Erlang VM speed at which it handles string operations is rather unimpressive. What is worse, since the XML handling libraries translate XML structures into Erlang records - which relay on the use of indivisible Erlang data atoms - the Erlang code required to handle a large XML object quickly grows into a huge block of repeated code [80]. Although there are ways of working with the immutable record names by the way of falling back to the Erlang tuple data types, this is a major limitation of the language and the immutability of the variables quickly strikes as a headache. The performance issues related to string processing can be resolved with the use of binary lists which Erlang processes really fast. Still there is the extra cost of getting to know this rather exotic data structure.

### 2.8.5 Heavy Computations

In general, since Erlang is a dynamic language, its codebase will be usually bigger than for a similar program written in C. It will therefore take more CPU cycles to execute the same calculation. This holds true to all dynamic languages. Also the running within a Virtual Machine imposes some restrictions as to the effective handling of memory. Java Virtual Machine is considered a better option for computation intensive applications, although its performance at number crunching tasks will naturally never match and equivalent program written in C.

### 2.8.6 Database Drivers

Although some noSQL database vendors such as Riak or CounchDB provide native Erlang drivers for their products, the SQL database vendors usually do not. The approach suggested by the OTP team is to use ODBC though this immediately brings back the problems related to close coupling of the Erlang VM with the OS scheduler managing multiple ODBC connections. Perhaps a more interesting option is relying on a tight coupling with a specific binary driver and creating a pool of processes within Erlang that make use of the lower level driver implementation therefore delegating the communication with the database to the driver still keeping the concurrency aspects within the Erlang VM. At the moment only PostgreSQL has a well tested and supported driver. Other popular databases such as Oracle, MSSQL or MySql are not well supported. Consequently there are no good ORM libraries for Erlang.

# 3 New Web Development Standards

## 3.1 Erlang in Web Development

Ruby on Rails, Python, Django, NodeJS and JavaEE are already proven and reliable ways of building interactive websites. All these technologies however have programming languages at their core that were not designed specifically for real-time, distributed network communications. Hot-code swapping, massive scaling and high availability are not part of their standard libraries. Even though these features can be somehow provided via external packages, programs and libraries, the resource cost of such added improvements is usually high, the quality often doubtful and the complexity of the code always increases. The most straight-forward example could be the JavaEE OSGI library [81] which provides some functionalities that Erlang OTP platform implements as standard. Erlang OTP applications have managed concurrency built in and the spawning and managing of processes is cheap. Ericsson built a language that is perfect for cheap communications hardware. Perhaps there are languages that provide better support for concurrency than Erlang. These languages however usually target specific hardware types, which are usually offered by specialised companies, at a higher price. Erlang provides exceptional performance on cheap commodity servers. This, along with Erlang regular and open-source releases, substantial community of developers and extensibility via Java and C interfaces make Erlang a natural choice for new start e-commerce businesses which provide an online services via a busy website at their core. Erlang has been academically researched in terms of its resilience, availability and dynamic reconfiguration [82], it is free, designed for distribution and networked environments and performs well on cheap and easily available hardware. It appears to be a natural choice for writing web applications, yet its adaptation for that purpose is rather slow. One of the main obstacles that Erlang needs to overcome in order to start achieving some popularity in the area is lack of unified set of tools that would enable fast development of Erlang web applications that meet the requirements of modern websites while making the best use of the standard Erlang features. Fortunately, the Erlang community continues to invest a lot of effort and the provision of an Erlang web application framework appears to be not far on the horizon. There are not any stable web application frameworks for Erlang available yet and one major obstacle that needs to be overcome is fact that there is no GUI IDE developed for Erlang in the first place. Coding in Erlang is done mostly in EMACS [83] though there are Eclipse [84] and IntelliJ [85] plug-ins available that augment the IDEs with the understanding of Erlang syntax. Even though there are not any stable Erlang web frameworks out there, developing web applications in Erlang is not an unpopular practise. O'Reilly published a handbook about web application development in Erlang [56], the Erlang Factories frequently feature presentations showing the advancements made in that area and there are Erlang libraries available that target the requirements of client - server communications.

## 3.2 Erlang Web Development Standards

Erlang OTP should be the default choice for anyone wanting to build a large, robust and extendable modern website relies on full duplex, bi-directional communication channels with the browsers. Helping to achieve this goal, the standards below built on key points that transpire as the most significant in the context of web application development that were discussed as part of the background survey. This chapter realises the first aim of this project.

### 3.2.1 OTP By Default

The main purpose for choosing Erlang as a web development tool is the desire to build a reliable and resilient system designed from the beginning with scaling in mind. The framework should guarantee all these qualities out of the box under condition the development is done with accordance to best OTP development practises.

### 3.2.2 Full Duplex Channel

Web applications built in Erlang will most likely spawn a new process for each incoming connection. Full-duplex, bi-directional communications will be a standard in web communications in the future. Erlang web applications should feature a good support for the management of a potentially very large number of concurrent HTTP connections, easily categorising, querying and distinguishing between them.

### 3.2.3 Solid Foundations

In order to stand a chance as an attractive alternative to other programming languages Erlang has to hold firmly to its guarantees and that is first and foremost to building scalable, highly available and distributed systems. These guarantees cannot be compromised by a poor design decision in one of the libraries that introduces a bottleneck and is difficult to debug or a lack of support for running in a distributed environment. A user of an Erlang web development framework should have every guarantee that the building blocks of the framework have been designed to work in a large OTP application avoiding bottlenecks and not compromising scaling abilities. Every core building block of the library should pass a series of well defined tests. These should include:

* the basic functionalities offered by the library
* the quality of code
* the support for running as part of a cluster
* the scaling abilities
* the levels of resiliency

At the same time it is important to give users the ability to easily replace any component of the system including the core libraries.

### 3.2.4 Pluggability

The building blocks of the framework should be pluggable and adding, removing and replacing of any dependencies should work in the standard Erlang OTP way. The framework should impose a minimum or no interfaces that the dependencies are required to implement. Having a rigidly defined and standardised plugin system would not only duplicate mechanisms already offered by OTP but also place an unnecessary constraints on developers.

### 3.2.5 Erlang and JavaScript

The framework should encourage coding in both Erlang for the back end and JavaScript for the front end. The approach of trying to shy away from Erlang clearly did not work. Certainly people who write Erlang have had some experience already with another language. JavaScript is not going to fade either and client side web development frameworks are really useful once you get to know them. There should be no attempt to encourage users to write front end code in Erlang. Instead the framework should ship with some core JavaScript libraries that the user may utilise to facilitate the communication with the Erlang backend. The framework should encourage the use of the existing client side web development frameworks such as AngularJS or Ember.

### 3.2.6 MVC

The application of a functional language to web application development and the changing nature of the web requires re-thinking some of the already adopted standards and conventions. The moving away from the generation of HTML on the server in favour of sending only data objects makes the MVC pattern no longer applicable for server side development. The application of MVC should be exercised within the client, leaving the server only with the processing and sending data.

### 3.2.7 Audience

Developing web application in Erlang should try to appeal to everyone but make no compromises and expect the user to invest time and effort into reading guides, following tutorials and most of all getting familiar with the challenges of building highly scalable applications. An average user of the framework should be well versed in OTP and feel comfortable editing both JavaScript and Erlang files.

# 4 Erlang Web Application Requirements

This section presents an overview of the most important requirements for a web application built as part of this project that realises the second aim of this paper. The set of requirements was formed in compliance with the standards presented in the last chapter. There will be no attempt to provide a definite list of requirements instead listing only those that are of key importance in the implementation of the previously formulated standards. It is important to note that due to the difficulties in assessing what is considered a good level of scaling or resilience some requirements have been arbitrarily defined. The requirements are categorised according to the MoSCoW framework [86].

## 4.1 Non-Functional Requirements

The web application must:

1. show scaling capabilities for simultaneous connections arriving at a pace of a few hundred per second
2. show resilience where a randomly dying processes in the VM does not crash the whole application
3. support all versions of HTTP protocol giving preference to bidirectional protocols encouraging HTTP/2 with fallbacks to older versions
4. use a JavaScript client side development framework library
5. support distributed deployments
6. have a pluggable architecture

The web application should:

1. give preference to binary data exchange format between client and server

## 4.2 Functional Requirements

The web application must:

1. register processes running long lived connections for full duplex protocols and provide a query-able process dictionary
2. map Erlang objects to SQL and NoSQL data
3. provide JavaScript libraries facilitating communication between the front end and back end
4. provide dynamically generated content

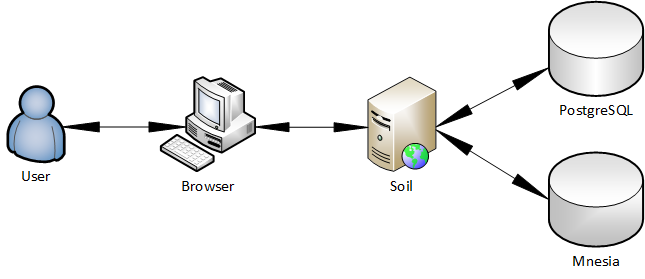
# 5 Implementation

This chapter introduces the Erlang application that was developed in view to meet the requirements outlined in previous chapter. This realises the second aim of this work.

## 5.1 High Level Architecture

The design of the solution follows the standard three-tier, client-server web application architecture which is depicted in . There will be front end Single Page Application client developed in JavaScript using one of the popular client-side development web framework libraries, the application server written in Erlang and a SQL (PostgreSQL) and noSQL (Mnesia) database providing a persistent storage for data.

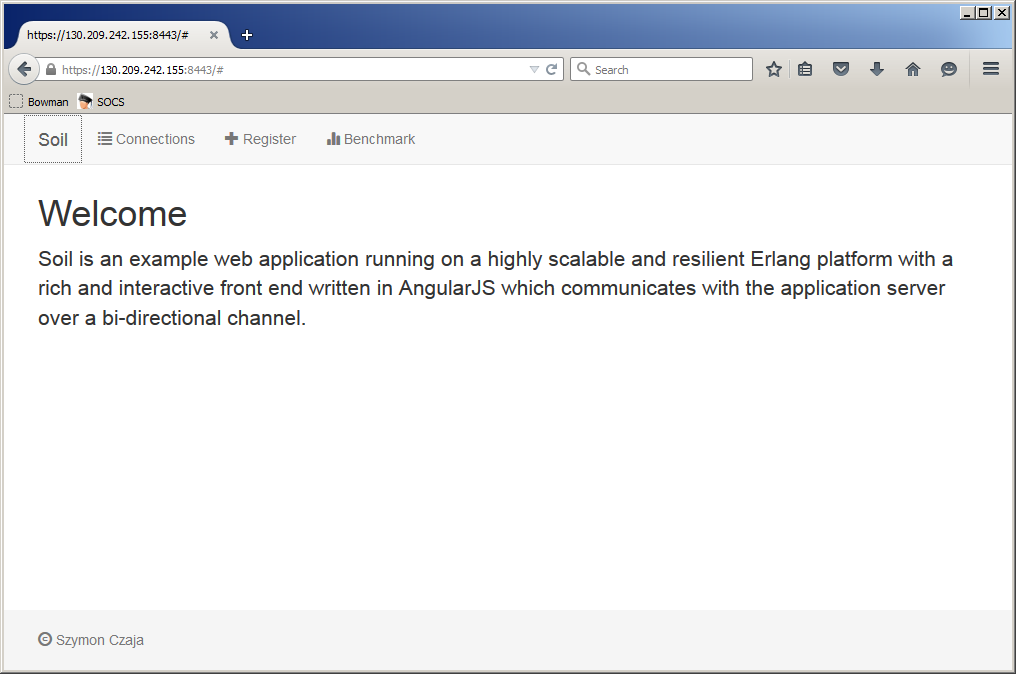
Figure : High level architecture



## 5.2 Client Application

The client application is developed in AngularJS. The client application has full ownership of the DOM within the browser. The processing of user events, the updating of views, the routing between the pages and maintenance of the client state is done internally within the application. The communication with the application server is facilitated using an internal AngularJS service which wraps up Bullet.js library which in turn heavily relies on jQuery. Appendix 4 represents the structure of the AngularJS project. The Bullet.js module in the services folder shown on the figure ensures websocket connectivity with Soil application server. The home page of the application is presented on Figure 2. The client side also features the 'Connections', 'Register' and 'Benchmark' tabs shown respectively in appendices 1, 2 and 3. The 'Connections' tab is used to show a list of live web socket sessions on the server. The 'Register' tab features a form which submits data to the server in JSON format. The data is mutated to Erlang maps and persisted to PostgreSQL database using the Norm library described in paragraph 5.4.1 Norm.

Figure : Soil home page



## 5.3 Application Server

The application server is an OTP application named 'Soil' [87] which was developed with view to meet the recommendations and requirements put forward in the preceding chapters. The server works with the client application trying to maintain an ongoing bi-directional, full duplex communication channel. Its main purpose is to provide and process data and ensure client communications remain secure. It works with a database in an efficient manner maintaining a pool of reusable connections using Norm library that was developed specifically for the purposes of the project.

### 5.3.1 Soil

Soil is the top-level component of the project. The application relies on a number of dependency libraries that provide the main functionalities. The list of core application dependencies is presented in appendix figure 4. Soil does not attempt to put any requirements on the interfaces that the dependent applications should implement instead attempting to use them in their original form. This approach guarantees the ease of extendibility though at times it might require some extra effort from the user to write additional code that would guarantee good interoperability. Soil uses Cowboy to bind to two TCP ports and start HTTP and HTTPS listeners. The Erlang code that starts the two listeners is presented in appendix item 6.

The application spawns its main supervisor process but does not create its own supervision tree relying instead on the process trees of its dependency applications. This greatly simplifies the program - pushing the burden of process management onto subordinate libraries. The application serves dynamically generated HTML content using ErlyDTL on the initiation of the web connection. The HTTP response which carries dynamically generated HTML page in its body also contains a cookie with a new session id in the header. The HTML is an index page which bootstraps the front end AngularJS client application. The client application initiates websocket connection which is handled in Soil by Cowboy using Bullet handler. Soil extracts a session id from a JSON object that the client application sent and uses Gproc to register the incoming connection with that session id. This is shown in appendix figure 7.

Gproc internally uses an in memory key-value database to store information about registered processes. The 'ets' database is shipped as standard with the Erlang VM and does not require any management overhead. The exchange of data between the client and server is carried out over a websocket connection exchanging JSON objects. The structure of JSON object consists of two key fields i.e. the header and the body fields. The header carries meta information that soil uses to route the object to the appropriate handler while the body has the actual data which usually represents some function arguments. Soil delegates all business logic handling to a single module for developer's convenience. The list of Soil modules with a number of lines of code per module is presented in appendix item 8.

## 5.4 Database

The building of a web application that realises the second aim of this project requires a careful choice of a SQL database. It has been discovered as part of the background survey that there are very few SQL drivers available for Erlang. It has been decided to use PostgreSQL mainly due to the availability of a well tested Erlang driver but also good scaling capabilities. Although the requirements of the project do not place high scaling requirements on the database and the main focus is to provide a convenient access to the SQL backend from Erlang using an ORM library, it has to be noted however that a production ready deployment requires a careful consideration mainly to ensure the database access does not become a bottleneck and meets the same quality requirements as the main application server.

### 5.4.1 Norm

Having decided on the SQL database and the implementation of the driver it quickly transpired that there was no ORM library that would conveniently map relational data to an Erlang data structure. The only considered candidate was the BossDB library developed for the ChicagoBoss project. It was decided to refrain from using it because of extensive use of parse transform and external dependencies which appeared to be an overkill for the purposes of the Soil project application. It was decided to start a new project due to the lack of suitable lightweight ORM library. The new project, named 'Norm' [88], was developed with the following goals:

* **Small** **codebase**

'Norm' aims to be have small code and does not spawn any process supervision trees on its own unless needed for pooling. The library aims to provide data mapping between database results and Erlang objects. The use of binaries and iolists is encouraged instead of lists or atoms. The list of Norm modules along with a total number of lines of code per module is presented in appendix figure 11.

* **Support of multiple databases**

The library aims to provide runtime support for a number of databases both SQL and NoSQL. Web applications usually need access to more than one database. SQL databases are great at storing highly normalised data that requires ACID properties, this type of databases is not however a good fit for fast lookup queries or BLOB retrieval. For these reasons the first version of Norm provides support for both PostgreSQL and Mnesia.

* **Relaxed unified interface**

Norm provides a unified interface to perform basic SAVE, FIND and REMOVE operations across all database types. While this is extremely convenient for developers, this may also be very constraining in practise. The saving of an object involves overwriting an existing object with the same key or inserting a new distinct record. In Mnesia database of 'set' type this is a single operation whereas in SQL these may require a SELECT query to check if object exists and then a separate INSERT or UPDATE. It would be extremely limiting if an ORM library restricted access to database specific operations. For this reason 'Norm' provides a unified interface only for a basic set of operations.

* **Extendibility**

Norm can be easily extended to add support for more databases. This involves writing an Erlang module that implements norm\_behaviour callbacks. The functionalities provided by the library can be easily enhanced with additional dependencies. In fact the first version of the library heavily rests on 'epgsql' library as a PostgreSQL driver and 'poolboy' library to provide PostgreSQL connection pooling.

* **JSON first**

Norm constructs objects as maps that can be easily converted to JSON objects using 'jsx' library. This means a single model definition for the front-end, the application server and the database. This is a major improvement over the currently adopted practise of mapping JSON to language specific classes and them mapping these class instances onto relational SQL format.

Norm uses an environment configuration variable which points to the module the library will read to fetch model definitions. These definitions which can be represented as Erlang maps or JSON objects, are used in initialisation functions of the library to create the required underlying stores i.e. tables in the case or SQL databases or buckets or indexes in case of noSQL. Appendix figure 9 presents mnesia model definitions and appendix no. 10 represents an example of model definitions for PostgreSQL. The definitions use Erlang maps although JSON format is also supported.

# 6 Evaluation

This section evaluates the project application in the context of the requirements it attempted to meet therefore allowing to draw conclusions about the capabilities of Erlang as a web development platform. This section also presents a set of recommendations formed as a result of the evaluation process therefore realising the third aim of this project.

## 6.1 Requirement Status

This section evaluates the implementation stage and outlines the status of the requirements met by the final project release.

### 6.1.1 Status of Non-Functional Requirements

#### 6.1.1.1 Requirement Testing

In order to determine the status of some non-functional requirements related to the scalability and resiliency of the Soil application it was necessary to employ third party testing tools. It was decided to use Tsung [89] for scalability testing and ChaosMonkey [90] for resiliency testing mainly due to their affinity with Erlang. The tests were performed on a virtualised Linux RHEL 6 server with 4 cores and 8 GB of RAM.

##### 6.1.1.1.1 Scalability Test Using Tsung

In order to simulate a scaling scenario it was necessary to create a large number of websocket connection initiating clients that would remain connected for some specified time. It was problematic to choose the number of clients that would attempt to connect to the server as it was not clear what level of scaling can be regarded as satisfactory thus it was arbitrarily decided to simulate a number of connection initiating clients which is large enough to cause Tsung to generate an error report.

###### 6.1.1.1.1.1 Test Settings

Tsung was set to use 64000 for the maxusers setting [91] which guaranteed to produce an error report. In order to create that many connections the file descriptor limit for the shell running Tsung was increased to 64000. The websockets connections were set to persist for 60 seconds before being terminated and the connection initiation rate was set at 1000 per second. The client arrival process was set to last 1 minute what guaranteed to reach a 60000 threshold. The Tsung xml settings file is presented in Figure 3.

Figure : Tsung Settings



###### 6.1.1.1.1.2 Test Results

Tsung test report presented in Figure 4 shows that Soil managed to respond to 47853 user initiated connections with 47520 being successfully connected. The test scenario generated 441 errors, 333 of which were error\_connect\_eaddrinuse and 108 error\_send\_einval. These are Tsung error codes [91] that use the standard POSIX error format [91] to report on the test results. The errors clearly point at problems with the scaling of the web application and therefore mark the scaling limit. The utilisation of hardware resources while running the test is presented in appendix figure 12.

Figure : Tsung results

|  |  |
| --- | --- |
| **Description** | **Results** |
| The main statistics table summarising key results | MainStats |
| The counter statistics table | counters_stats |
| The network throughput table | networks_throughput |
| The errors table | errors |

##### 6.1.1.1.2 Resilience Test Using Tsung And ChaosMonkey

The purpose of the resilience test is to check whether Erlang VM process running Soil application remains responsive after killing a random process within the system.

###### 6.1.1.1.2.1 Test Settings

Since a detailed resiliency test was neither a goal of this project nor was it clearly defined in the requirements for the soil application, the resiliency testing was conducted on top of scalability testing re-using scalability test settings and adding ChaosMonkey which was started prior to the launching of Tsung with the chaos\_monkey:almost\_kill() [91] command.

###### 6.1.1.1.2.2 Test Results

The figures compiled by Tsung are very similar to the results reported for the scalability test. Soil acknowledged 48193 user initiated connections which is actually more than in the previous test, successfully connecting 47520 which is exactly as many as in the scalability scenario. The running of the ChaosMonkey however resulted in more errors reported during testing what was to be expected. The total number of errors totalled 851. The results of the test are presented in .

Figure : Statistics report of Tsung scalability test results with Chaos Monkey running

|  |  |
| --- | --- |
| **Description** | **Results** |
| The main statistics table summarising key results | mainStats |
| The counter statistics table | counter |
| The network throughput table | networkThroughput |
| The errors table | errors |

#### 6.1.1.2 Requirement Status

The Table 1 presents the status of the non-functional requirements from section 4.1 Non-Functional Requirements.

Table : Status of Non-Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Priority** | **Requirement** | **Status** | **Comments** |
| Must | show scaling capabilities for simultaneous connections arriving at a pace of a few hundred per second | Met\* | The application serviced 47520 connected clients in the space of 60 seconds. |
| Must | show resilience where a randomly dying processes in the VM does not crash the whole application | Met\* | The application remained responsive after killing a number of random processes in the system. |
| Must | support all versions of HTTP protocol giving preference to bidirectional protocols encouraging HTTP/2 with fallbacks to older versions | Not Met | No existing library that would provide full HTTP/2, SPDY and Websockets implementation was found. |
| Must | use a JavaScript client side development framework library | Met | The client was implemented in AngularJS. |
| Must | support distributed deployments | Partly Met | Although the code of Soil was written without much scope for distributed deployment, this is partly provided by the Erlang system and there would not be much effort needed to ensure Soil fully support it. |
| Must | have a pluggable architecture | Met | Soil is a standard OTP application that can be easily extended with additional application libraries. |
| Should | give preference to binary data exchange format between client and server | Partly Met | Soil internally uses binary extensively, more effort is required however to ensure binary data is exchanged between the client and the server. |

\* Please refer to 7.2 Limitations.

### 6.1.2 Status of Functional Requirements

The Table 2 presents the status of the functional requirements from section 4.2 Functional Requirements.

Table : Status of Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Priority** | **Requirement** | **Status** | **Comments** |
| Must | register processes running long lived connections for full duplex protocols and provide a query-able process dictionary | Met | Gproc library provides a robust way of registering processes using a fast in-memory database and communicating between them. |
| Must | map Erlang objects to SQL and NoSQL data | Met | Norm library was developed for the purpose of mapping JSON objects to Erlang maps which the library later converts into respective SQL statements or Mnesia records. |
| Must | provide JavaScript libraries facilitating communication between the front end and back end | Met | AngularJS service module was developed which facilitates communication between the client and the server. |
| Must | provide dynamically generated content | Met | ErlyDTL library is used to provide dynamically generated content |

## 6.2 Recommendations

### 6.2.1 Libraries

Erlang community needs more libraries of better quality. There are some libraries of great quality that were developed by Erlang enthusiasts however in order to guarantee continued support there should be an ongoing endeavour to gain involvement of community members who use Erlang professionally to develop applications for production environments. Although there is a number of applications written in Erlang that have a large user base Erlang still remains a niche language which is rarely considered as a base platform in enterprises.

#### 6.2.1.1 Popularising Erlang

##### 6.2.1.1.1 Academia

One way to popularise Erlang is to teach it at university. There is already a number of institutions around the world that provide courses that aim to introduce students to the concepts of functional programming using Erlang. It is difficult to assess what impact the using of Erlang in academia has on the overall popularity of the language surely however the using of Erlang purely to demonstrate its behaviour as a functional language does not have much to do with the popularising of the OTP platform as possibly the most reliable way of building soft-real time systems.

##### 6.2.1.1.2 Object-Orientating

The functional nature of the language which is the very source of the guarantees delivered by the OTP platform absurdly is also the reason of the Erlang's lack of popularity. There is not probably a single functional language which popularity would match any of the main stream object oriented languages. Scala owes its success to the compatibility with JVM and it is unlikely it would gain so much popularity if it was not seen as a cousin of Java. Considering recent C# and Java enhancements there appears to be a drive to introduce functional way of writing code into object oriented languages rather that to decouple the two [92]. Interestingly there are discussions as to what object-orientation actually means and there are voices implying Erlang is indeed an object oriented language [93]. An assessment of whether Erlang is object-oriented is beyond the scope of this project, certainly however Erlang has not implemented characteristics that the majority of the main stream object-oriented language currently exhibit. It is therefore suggested some further research is needed to assess whether Erlang has any room to accommodate some of the characteristics that are currently regarded as attributes of an object-oriented language for the purpose of making the language more familiar to an average software developer but also making the platform more functional.

#### 6.2.1.2 Promoting Best Practises

The promoting of the best practises to develop good quality Erlang code can be seen as a natural consequence of popularising Erlang as a coding language. The promoting of the good practises however can also be regarded in isolation i.e. in the current Erlang community landscape where discussions about the language and most of the development is done primarily by enthusiasts rather than experts. The open source Erlang community need vehicles that would promote the good practise of the code they are developing. Unfortunately most of the best practise in Erlang development comes from an exposure to other languages, Erlang specific best practises are shared only around experienced Erlang users.

### 6.2.2 Erlang ORM

The difficulty in writing a good ORM library for Erlang stems from the fact it is a functional language which advocates the concept of immutability and from a lack of good SQL database drivers for Erlang.

#### 6.2.2.1 Immutability

On one hand this greatly simplifies coding massively concurrent application as each process is given its own chunk of memory, on the other hand however the immutability of variables makes it extremely hard to work with dynamic structures.

##### 6.2.2.1.1 Classes

Object-oriented languages have a concept of class which usually represents a named object with a set of properties that can be easily mutated. Erlang does not have anything like a class, instead it uses modules which do not have any form of globally shared memory space therefore no properties. Any attempt to represent an object using a module resolves to a parse transform i.e. generating and compiling module code at runtime which is very slow and in general discouraged. This approach to representing relational data was adopted in boss\_db library which was a cornerstone of the Chicago Boss project. Erlang has however a data structure that is similar to an object i.e. a record.

##### 6.2.2.1.2 Records

Records have names and a set of user defined properties. They are defined usually in an external file with 'hrl' extension which is declared in every module that needs to use the defined records. Records however are only syntactic sugar which is supposed to save the programmer some time writing a key-value list lookup code at the development time. Records are translated to tuples at compile time, losing any information about property names thus property based access is simply not possible at runtime. The static nature of records along with their internal representation as tuples makes them a rather unsuitable candidate for ORM data representation.

##### 6.2.2.1.3 Maps

A strong demand for a new property based dynamic data structure has finally been addressed in the 17 release of the Erlang OTP. The new version of Erlang runtime introduced maps which support named properties that can be declared and mutated at runtime. Unfortunately maps cannot be named and are more similar to JSON rather than objects known from object-oriented languages. The Norm library that has been developed for the Soil project uses maps extensively adopting a novel approach of converting Erlang maps to JSON objects. The reliance on JSON however requires object metadata to be stored as an extra field in every object. For the purpose of an ORM library the metadata stores information about the object name along with the types of variables that the object fields are referencing. Named maps or dynamic run-time records would be a much better fit to represent ORM data.

#### 6.2.2.2 Drivers

Among all SQL databases only PostgreSQL and MySQL are satisfactorily well supported in Erlang and are suitable for production use. It is extremely difficult to find a good driver for MSSQL or Oracle SQL database. There is a number of community projects that aim to deliver Oracle database interface to Erlang though unfortunately there appears to be no real drive for the database vendors who currently are the main market players to provide drivers for Erlang. This greatly constraints the use of Erlang in production environments. There should be a serious and persevering effort within the community to rectify this.

### 6.2.3 Code Repository

Most popular programming languages have code repositories e.g. Java has Maven, JavaScript has npm and Perl has CPAN. Erlang does not have one instead most of the projects are hosted on Git and Subversion servers. This is not surprising considering a rather niche nature of the language. Erlang community however may wish to invest efforts into creating a code repository that would perhaps be accompanied by a release tool. Iterating over all the benefits of having a code repository is beyond the scope of this project. It is important to emphasise though that the development of such a repository would constitute an important part in the efforts of popularising Erlang.

### 6.2.4 Release Tool

Erlang OTP features a release tool application [94] which helps to package Erlang code along with any dependencies into a single, standalone release package that can be run without the need of having Erlang installed natively on the target system. The tool however has not gained much adoption among the members of the Erlang community mainly because its limited support for the fetching of external dependencies and rather cumbersome use. The need for an easily usable release tool that would meet the practical requirements of an Erlang developer was answered by a company which uses Erlang for the development of their flagship product i.e. Riak database. Basho released their packaging tool called rebar to wider community as open source. The tool became a standard utility used by Erlang developers to compile and package Erlang applications. Rebar relies on reltool heavily and although it greatly simplifies the compilation and packaging process it is far from perfect. Rebar does not detect dependencies that do not require recompilation and its support for compiling C extensions is very limited. The recommendation for building a better release tool is being addressed already. In an effort to develop a better tool a new project has been spun off in an attempt to develop a better version of rebar named rebar3 [95]. At the time of writing the project is in active development.

### 6.2.5 Testing

Erlang community already has a range of good testing tools e.g. EUnit [96] and Proper [97]. These however test only Erlang code. What Erlang applications would really benefit from is a standard for scalability and resilience testing. There are tools that can already facilitate such testing such as Tsung and ChaosMonkey. There is however no standard approach to such testing. Developing a standard for assessing the scalability and resilience levels of an Erlang application would give Erlang something novel that other web development platforms would aspire to either coin on their own or indeed borrow from Erlang.

# 7 Conclusion

## 7.1 Summary

Erlang web development should not try to replicate the existing practises and conventions adopted in the industry. They were shaped at the time by the limits of the HTTP protocol, client side technologies and programming practises popularised by object-oriented development languages. Instead Erlang should establish its own standards, provide a high quality and elegant implementation and demonstrate that these new standards are a better fit for the web development challenges of the future websites. The background survey showed that the publication of the specification of HTTP/2.0 opens a new chapter in the history of web application development. This may be the one opportunity for Erlang OTP to mark its presence as a web development framework as the previous attempts clearly did not work. The presence of JavaScript client side development frameworks is becoming more and more apparent, and the industry is clearly moving away from the compilation of HTML on the server side. The attempts of trying to write parsers that would translate Erlang code into JavaScript have failed to attract wider audience. Similarly the projects which hid the Erlang implementation from the developer did not achieve much success. Popularisation of new web development standards must go hand in hand with code implementation, and it is precisely the libraries that will aid the development of future websites which Erlang OTP is currently missing. A well designed ORM library would constitute a very valuable contribution. Perhaps a continuing improvement of the Norm library would guarantee sufficient thrust to provide the community with a convenient, flexible and reliable way of accessing databases from Erlang that would eventually translate into a standard. Interestingly, the guarantees of web applications, that Erlang OTP is suited to deliver, have been directly targeted by the Play framework in what was called the Reactive manifesto [98]. In fact, the key features of a reactive system that the manifesto outlines, are the exact features that describe applications built in Erlang OTP. It is disappointing that other languages manage to reinvent and successfully popularise ideas which were present in Erlang OTP for years. This only confirms that the design decisions made by the OTP team were correct and the chances of Erlang or indeed Elixir gaining more popularity are now higher than ever.

## 7.2 Limitations

The scalability and resiliency tests were greatly simplified mainly because there was no attempt to formulate detailed quality criteria for these tests earlier. There is no official standard set for a minimum scalability level of a web application so the testing scenario had to be devised in an arbitrary manner. There was no attempt to validate Tsung results nor drill into finer details. It is not exactly clear how much damage ChaosMonkey inflicted while running the resiliency tests either. The value of 1000 websocket initiating web requests per second simulated by Tsung was chosen using trial and error method until Tsung generated an error report thus marking the scalability capabilities of the tested application. It is unclear whether the chosen value truly represents the scaling capabilities of the benchmarked application server. Further inquiry would have to be made into the settings of the test environment and the state of Linux system at the time of the test. These facts however should not be regarded as disruptive in any sense - they rather serve to provide more depth in the evaluation of the requirements.

## 7.3 Further Work

This section provides some suggestions for further work required to make Erlang effectively a popular tool for web application development. The suggestions for further work presented here encompass different domains and range from the recommendations for making improvements to the Soil and Norm libraries to the promoting of certain practises within wider Erlang community of web application developers.

### 7.3.1 Further Work On This Project

This project would benefit the most from better scalability and resiliency tests. In order to have better testing some research would have to be carried out in order to determine what level of scalability should be expected from an average web application. A similar research would have to be conducted in the field of resiliency testing. The findings of such research would help to define more specific Erlang web application requirements the meeting of which would ultimately deny or confirm the superiority of the scaling abilities of the platform over the technologies that currently dominate the market.

#### 7.3.1.1 Improving Soil

Soil AngularJS Bullet service could be improved to remove reliance on jQuery library and depend solely on AngularJS. The Soil application could in general use a better selection of the client side connection handling plugins that would support different JavaScirpt client side frameworks such as Ember or ExtJS. There is a lot room for improvement both in the selection of client side libraries as well as their quality. One of the biggest quality improvements would be the adding of support for binary data exchange format e.g. BERT. Serializing JSON objects into a binary format with a smaller memory footprint, exchanging binary encoded data over the network and converting back on the server side into JSON would bring an improvement in the user experience without sacrificing developer's convenience.

#### 7.3.1.2 Improving Norm

Norm needs adding more support for the databases it currently targets along with writing more test scenarios. In general the library needs work to ensure it works in a robust manner. What would Norm certainly require over time would be extra modules that provide interaction with other databases.

### 7.3.2 Further Improvements To Erlang

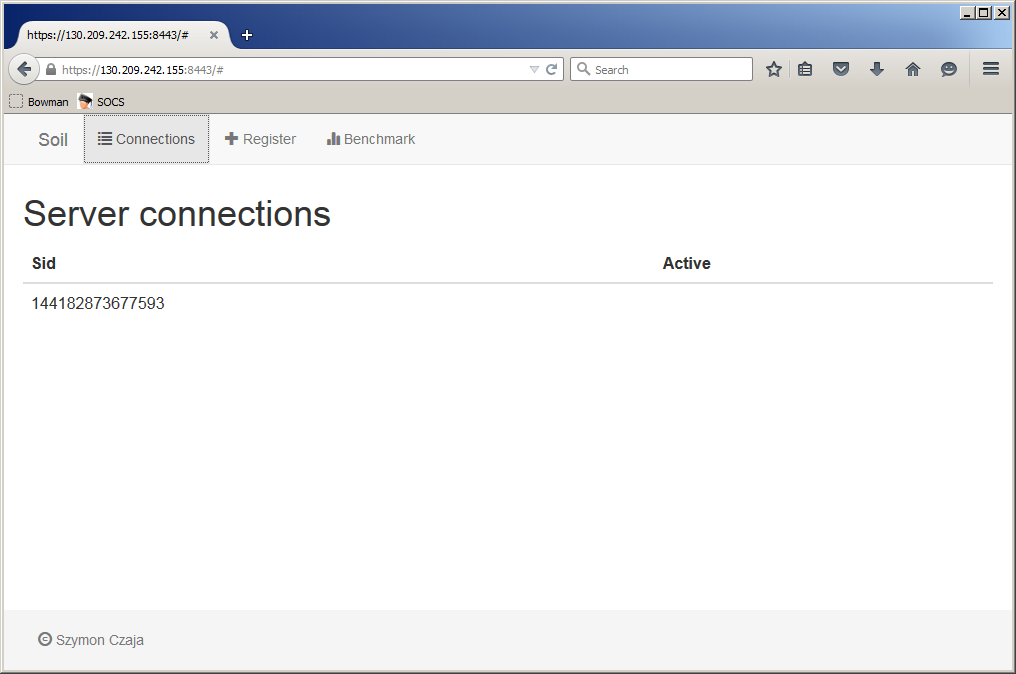
The creators of Erlang might gain a great deal from research on how the platform could be possibly improved with characteristics that are associated with object-oriented development. There is certainly a trend of extending the capabilities of object-oriented languages with functional expressions. Perhaps it is worth finding out if Erlang could equally gain from materialising concepts within the language that are usually associated with object-oriented languages. One way of making Erlang more familiar to an object-oriented programmer and therefore an average developer might be by the way of introducing a dynamic record or a named map data structure into Erlang which would allow those starting with the language to use a data type they are likely to associate with data objects they have had experience of working with. Finally Erlang would gain a big improvement if the biggest database vendors started shipping Erlang drivers for their products. This is however very unlikely to happen in a short run.

## 7.4 Challenges For The Erlang community

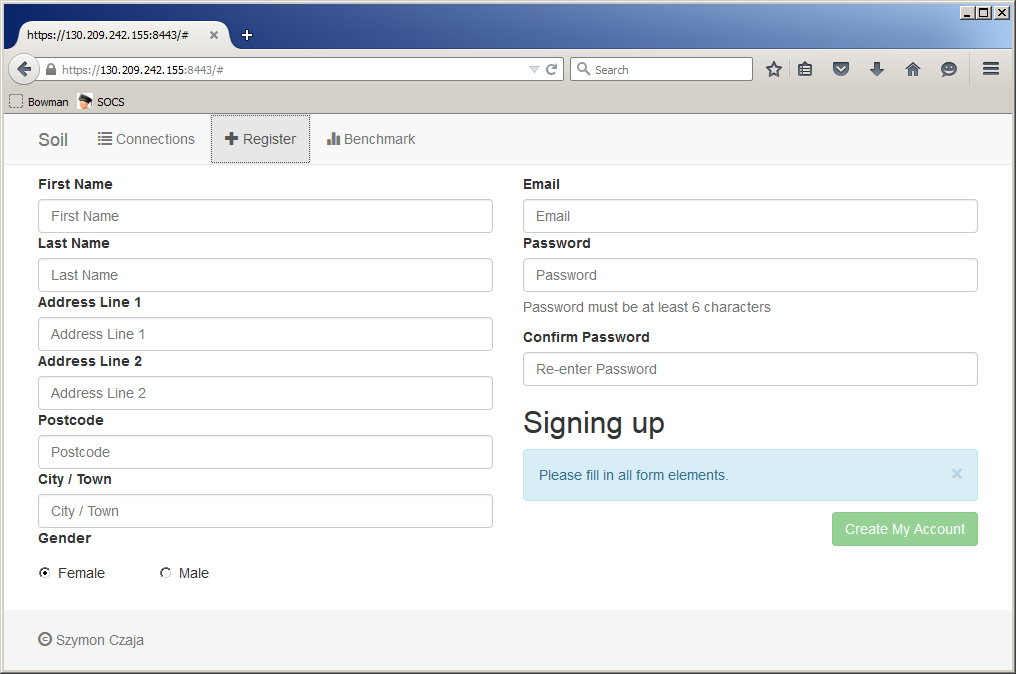
Erlang community needs more libraries of better quality. Since most Erlang libraries are OTP applications what would be really beneficial is a standard test that would benchmark the application in terms of its scaling abilities and resiliency. Erlang programmers would gain a lot from some form of standardisation for scalability and resilience testing. This would be something novel that other web application community currently do not entertain.

# Appendix A - List Of Figures

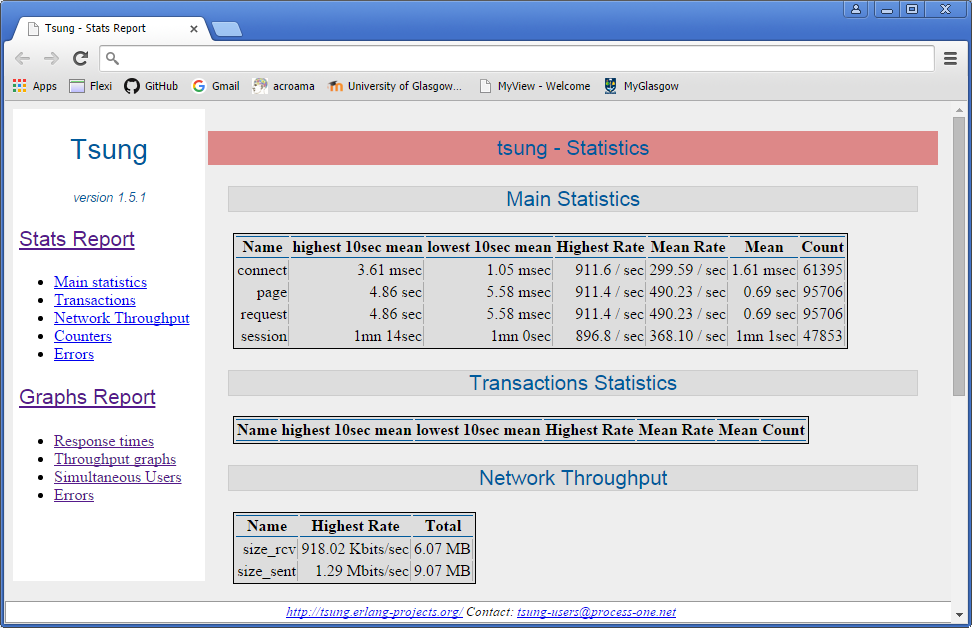
1. **Connections tab**



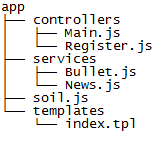
1. **Register tab**

****

1. **Benchmark tab**



1. **AngularJS project structure**

****

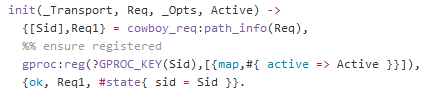
1. **Soil dependency applications**

|  |  |
| --- | --- |
| **Application** | **Description** |
| jsx | JSON parser capable of converting JSON utf16 binary to Erlang maps |
| cowboy | HTTP server supporting HTTP 1.1, HTTPS and Websockets |
| bullet | Cowboy server side handler for always connected client |
| gproc | Extended process dictionary |
| norm | ORM library mapping JSON to Erlang maps |
| erlydtl | Django DSL template language for dynamic HTML rendering |

1. **HTTP and HTTPS listener start**



1. **New connection initialization and Gproc registration**



1. **Soil modules**

|  |  |  |
| --- | --- | --- |
| **Module** | **Description** | **LOC** |
| soil.app.src | Erlang application .app file | 12 |
| soil\_app.erl | Main OTP application module | 73 |
| soil\_sup.erl | Main OTP supervisor module | 28 |
| soil.erl | Business logic module | 57 |
| soil\_bullet\_handler.erl | Implements Bullet callbacks to handle incoming connections | 40 |
| soil\_db.erl | Defines database access specific function | 6 |
| soil\_models.erl | Defines model definitions used by Norm application | 64 |
| soil\_rest.erl | Soil REST handler | 46 |
| soil\_session.erl | Defines HTTP session specific modules | 18 |
| soil\_utls.erl | Defines common functions | 47 |
|  |  | 391 |

1. **Mnesia model definition**



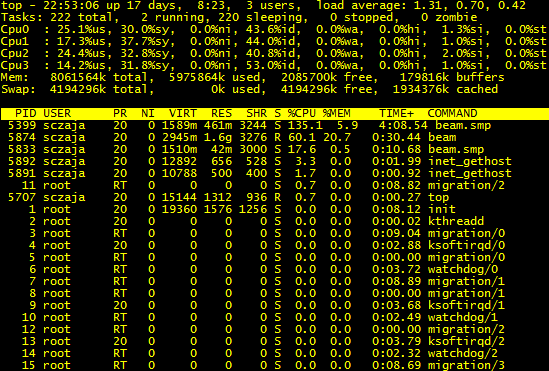
1. **PostgreSQL models definitions**



1. **Norm modules**

|  |  |  |
| --- | --- | --- |
| **Module** | **Description** | **LOC** |
| norm.app.src | Erlang application .app file | 12 |
| norm\_app.erl | Main OTP application module | 43 |
| norm\_sup.erl | Main OTP supervisor module | 52 |
| norm\_behaviour.erl | Interface module defining a set of callback functions | 22 |
| norm.erl | Provides a set of cross database functions | 81 |
| norm\_mnesia.erl | Mnesia interaction handling module | 280 |
| norm\_pgsql.erl | PostgreSQL interaction handling module | 719 |
| norm\_pgsql\_worker.erl | A worker gen\_server for creating connection pool | 43 |
| norm\_models.erl | Sample model definition file | 51 |
| norm\_utls.erl | Defines common functions | 223 |
| norm\_log.erl | Defines logging functions | 18 |
|  |  | 1544 |

1. **Output of top command while running Tsung**



x

x

x

x

x

x

x

x

x

x

x

# Bibliography

x

|  |  |
| --- | --- |
| [1] | Nottingham M. mnot’s blog. [Online].; 2015 [cited 2015 July 1]. Available from: <https://www.mnot.net/blog/2015/02/18/http2>. |
| [2] | erlang.org. Erlang Enhancement Proposal 43: Maps. [Online].; 2015 [cited 2015 March 1]. Available from: <http://www.erlang.org/eeps/eep-0043.html>. |
| [3] | Kohavi R, Longbotham R, Walker T. Online Experiments: Lessons Learned. IEEE Computer. 2010 September; 43. |
| [4] | Google. Google. [Online].; 2015 [cited 2015 May 10]. Available from: <https://developers.google.com/speed/spdy/?hl=en>. |
| [5] | NodeJS. NodeJs. [Online].; 2015 [cited 2015 August 25]. Available from: <https://nodejs.org/en/>. |
| [6] | International E. http://www.ecma-international.org. [Online].; 2015 [cited 2015 July 15]. Available from: <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>. |
| [7] | Kiran R. http://www.sitepoint.com. [Online].; 2015 [cited 2015 July 15]. Available from: <http://www.sitepoint.com/writing-angularjs-apps-using-es6/>. |
| [8] | stefanpenner. GitHub. [Online].; 2015 [cited 2015 July 15]. Available from: <https://github.com/stefanpenner/ember-app-kit>. |
| [9] | BSON. bsonspec.org. [Online].; 2015 [cited 2015 July 5]. Available from: <http://bsonspec.org/>. |
| [10] | Apache. Thrift Apache. [Online].; 2015 [cited 2015 July 15]. Available from: <https://thrift.apache.org/>. |
| [11] | Google. Protocol Buffers. [Online].; 2015 [cited 2015 July 15]. Available from: <https://developers.google.com/protocol-buffers/>. |
| [12] | Preston-Werner T. Bert RPC. [Online].; 2015 [cited 2015 July 1]. Available from: <http://bert-rpc.org/>. |
| [13] | Django. Django Templating. [Online].; 2015 [cited 2015 July 16]. Available from: <https://docs.djangoproject.com/en/1.8/topics/templates/>. |
| [14] | Oracle. Oracle. [Online].; 2015 [cited 2015 July 16]. Available from: <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>. |
| [15] | Spring. Spring. [Online].; 2015 [cited 2015 July 20]. Available from: <https://spring.io/>. |
| [16] | Friesen J. Java World. [Online].; 2016 [cited 2015 July 16]. Available from: <http://www.javaworld.com/article/2078848/java-concurrency/java-concurrency-java-101-the-next-generation-java-concurrency-without-the-pain-part-2.html>. |
| [17] | Apache. Apache Velocity. [Online].; 2015 [cited 2015 July 17]. Available from: <https://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html>. |
| [18] | Freemaker. Freemaker. [Online].; 2015 [cited 2015 July 16]. Available from: <http://freemarker.org/>. |
| [19] | JBoss. RichFaces. [Online].; 2015 [cited 2015 May 10]. Available from: <http://richfaces.jboss.org/>. |
| [20] | OpenFaces. OpenFaces. [Online].; 2015 [cited 2015 May 10]. Available from: <http://openfaces.org/>. |
| [21] | Soft M. Mule Soft. [Online].; 2015 [cited 2015 July 16]. Available from: <http://www.mulesoft.org>. |
| [22] | JBoss. JBoss. [Online].; 2015 [cited 2015 July 16]. Available from: <https://www.jboss.org/overview/>. |
| [23] | Django. Django Project. [Online].; 2015 [cited 2015 July 20]. Available from: <https://www.djangoproject.com/>. |
| [24] | Django. Django Databases. [Online].; 2015 [cited 2015 May 10]. Available from: <https://docs.djangoproject.com/en/1.8/ref/databases/>. |
| [25] | Django. Django. [Online].; 2015 [cited 2015 July 20]. Available from: <http://www.quora.com/What-is-the-technology-stack-behind-Pinterest-1>. |
| [26] | Instagram-engineering. Tumblr. [Online].; 2015 [cited 2015 July 20]. Available from: <http://instagram-engineering.tumblr.com/post/13649370142/what-powers-instagram-hundreds-of-instances>. |
| [27] | Symfony. Symfony. [Online].; 2015 [cited 2015 July 21]. Available from: <https://symfony.com/>. |
| [28] | Doctrine. Doctrine. [Online].; 2015 [cited 2015 July 22]. Available from: <http://www.doctrine-project.org/projects/orm.html>. |
| [29] | Symfony. Symfony. [Online].; 2015 [cited 2015 July 20]. Available from: <http://symfony.com/blog/delicious-preview-built-with-symfony>. |
| [30] | Symfony. Symfony. [Online].; 2015 [cited 2015 July 20]. Available from: <http://symfony.com/blog/yahoo-bookmarks-uses-symfony>. |
| [31] | Wage JH. joind. [Online].; 2015 [cited 2015 July 20]. Available from: <https://joind.in/8008>. |
| [32] | Rails. Ruby on Rails. [Online].; 2015 [cited 2015 July 20]. Available from: <http://rubyonrails.org/>. |
| [33] | Finley K. readwrite. [Online].; 2011 [cited 2015 July 20]. Available from: <http://readwrite.com/2011/07/06/twitter-java-scala>. |
| [34] | mojobo. GitHub. [Online].; 2009 [cited 2015 July 20]. Available from: <https://github.com/blog/530-how-we-made-github-fast>. |
| [35] | Tom Mornini EY. businessinsider. [Online].; 2011 [cited 2015 July 20]. Available from: <http://www.businessinsider.com/heres-why-ruby-on-rails-is-hot-2011-5?IR=T>. |
| [36] | Framework P. Play Framework. [Online].; 2015 [cited 2015 July 20]. Available from: <https://www.playframework.com/>. |
| [37] | Scala. scala-lang. [Online].; 2015 [cited 2015 July 20]. Available from: <http://www.scala-lang.org/>. |
| [38] | Hewitt C, Bishop P, Steiger R. A universal modular actor formalism for artificial intelligence. In 3rd International Joint Conference on Artificail Intelligence; 1973; San Francisco: Morgan Kaufman Publishers Inc. p. 234-235. |
| [39] | Humble C. InfoQ. [Online].; 2011 [cited 2015 July 20]. Available from: <http://www.infoq.com/articles/guardian_scala>. |
| [40] | AngularJS. AngularJS. [Online].; 2015 [cited 2015 July 22]. Available from: <https://angularjs.org/>. |
| [41] | Cesarini SaTF. Erlang programming. 1st ed.: O'Reilly Media, Inc.; 2009. |
| [42] | OTP E. erlang.org. [Online].; 2015 [cited 2015 May 20]. Available from: <http://www.erlang.org/doc/man/dialyzer.html>. |
| [43] | Eckel B. Thinking in java. 4th ed.: Prentice Hall; 2006. |
| [44] | OTP E. Erlang. [Online].; 2015 [cited 2015 July 26]. Available from: <http://erlang.org/doc/man/inets.html>. |
| [45] | ninenines. http://ninenines.eu/. [Online].; 2015 [cited 2015 January 21]. Available from: <https://github.com/extend/cowboy>. |
| [46] | klacke. GitHub. [Online].; 2015 [cited 2015 January 21]. Available from: <https://github.com/klacke/yaws>. |
| [47] | mochi. GitHub. [Online].; 2015 [cited 2015 January 21]. Available from: <https://github.com/mochi/mochiweb>. |
| [48] | Basho. Basho. [Online].; 2015 [cited 2015 January 21]. Available from: <http://basho.com/tag/webmachine/>. |
| [49] | Ostinelli R. http://www.ostinelli.net. [Online].; 2011 [cited 2015 February 14]. Available from: <http://www.ostinelli.net/a-comparison-between-misultin-mochiweb-cowboy-nodejs-and-tornadoweb/>. |
| [50] | Vinoski S. Steve Vinoski's Blog. [Online].; 2011 [cited 2015 February 14]. Available from: <http://steve.vinoski.net/blog/2011/05/09/erlang-web-server-benchmarking/>. |
| [51] | mochi. GitHub. [Online].; 2015 [cited 2015 February 14]. Available from: <https://github.com/mochi/mochiweb/blob/master/README>. |
| [52] | ErlyDTL. ErlyDTL. [Online].; 2015 [cited 2015 March 1]. Available from: <https://github.com/erlydtl/erlydtl>. |
| [53] | Nitrogen. GitHub. [Online].; 2015 [cited 2015 March 1]. Available from: <https://github.com/nitrogen/simple_bridge>. |
| [54] | JQuery. JQuery. [Online].; 2015 [cited 2015 July 20]. Available from: <https://jquery.com/>. |
| [55] | EarlyORM. GitHub. [Online].; 2015 [cited 2015 March 1]. Available from: <https://github.com/ErlyORM/boss_db>. |
| [56] | Kessin Z. Building Web Applications with Erlang. 1st ed.: O'Reilly Media; 2012. |
| [57] | ninenines. GitHub. [Online].; 2015 [cited 2015 March 1]. Available from: <https://github.com/extend/bullet>. |
| [58] | Preston-Werner T. Github. [Online].; 2009 [cited 2014 April 1]. Available from: <https://github.com/blog/531-introducing-bert-and-bert-rpc>. |
| [59] | HyperNumbers. GitHub. [Online].; 2014 [cited 2014 April 6]. Available from: <https://github.com/hypernumbers/luvviescript>. |
| [60] | SynRC. SynRC. [Online].; 2014 [cited 2015 April 6]. Available from: <https://github.com/synrc/kvs>. |
| [61] | ninenines. ninenines. [Online].; 2013 [cited 2015 March 1]. Available from: <http://ninenines.eu/talks/farwest/farwest.html>. |
| [62] | ninenines. GitHub. [Online].; 2013 [cited 2015 March 1]. Available from: <https://github.com/extend/farwest/wiki/2013-Fundraiser>. |
| [63] | Zotonic. Zotonic. [Online].; 2014 [cited 2015 April 1]. Available from: <http://zotonic.com/showcase>. |
| [64] | Drupal. Drupal. [Online].; 2015 [cited 2015 April 1]. Available from: <https://www.drupal.org/>. |
| [65] | Joomla. Joomla. [Online].; 2015 [cited 2015 April 1]. Available from: <https://www.joomla.org/>. |
| [66] | Wordpress. Wordpress. [Online].; 2015 [cited 2015 April 1]. Available from: <https://wordpress.com/>. |
| [67] | Miller E. Google. [Online].; 2013 [cited 2015 April 2]. Available from: <https://groups.google.com/forum/#!topic/chicagoboss/ekU5gZ_Ty2o>. |
| [68] | Cogswell J. dice.com. [Online].; 2014 [cited 2015 April 2]. Available from: <http://insights.dice.com/2014/10/09/5-programming-languages-marked-for-death/>. |
| [69] | Hao BTW. SitePoint. [Online].; 2013 [cited 2015 April 2]. Available from: <http://www.sitepoint.com/elixir-love-child-ruby-erlang/>. |
| [70] | Kehoe D. RailsApps. [Online].; 2013 [cited 2015 April 2]. Available from: <http://railsapps.github.io/what-is-ruby-rails.html>. |
| [71] | Editors BT. http://bigthink.com. [Online].; 2010 [cited 2015 April 2]. Available from: <http://bigthink.com/the-voice-of-big-think/rails-creator-david-heinemeier-hansson-explains-why-he-loves-ruby>. |
| [72] | Trainer K. http://erlang.org. [Online].; 2013 [cited 2014 May 7]. Available from: <http://erlang.org/pipermail/erlang-bugs/2013-October/003775.html>. |
| [73] | Andin I. erlang.org. [Online].; 2013 [cited 2014 May 7]. Available from: <http://erlang.org/pipermail/erlang-questions/2013-June/074381.html>. |
| [74] | Andin I. erlang.org. [Online].; 2014 [cited 2014 May 7]. Available from: <http://erlang.org/pipermail/erlang-questions/2014-April/078537.html>. |
| [75] | Miller E. evanmiller. [Online].; 2012 [cited 2014 May 22]. Available from: <http://www.evanmiller.org/why-i-program-in-erlang.html>. |
| [76] | Eralng. erlang.org. [Online].; 2015 [cited 2015 May 22]. Available from: <http://www.erlang.org/doc/apps/erts/>. |
| [77] | ninenines. ninenines. [Online].; 2014 [cited 2014 June 5]. Available from: <http://ninenines.eu/articles/erlang-scalability>. |
| [78] | mochi. GitHub. [Online].; 2014 [cited 2015 February 20]. Available from: <https://github.com/mochi/mochiweb/blob/master/src/mochijson2.erl>. |
| [79] | willemdj. GitHub. [Online].; 2015 [cited 2015 February 20]. Available from: <https://github.com/willemdj/erlsom>. |
| [80] | Inaimathi. StackOverflow. [Online].; 2012 [cited 2015 February 20]. Available from: <http://stackoverflow.com/questions/10821930/erlang-dynamic-record-editing>. |
| [81] | OSGI. OSGI. [Online].; 2015 [cited 2015 Febraury 20]. Available from: <http://www.osgi.org/Download/HomePage>. |
| [82] | Trinder P, King P, Nystrom JH. High-level distribution for the rapid production of robust telecoms software: Comparing C++ and Erlang. Edinburgh: Heriot-Watt University; 2013. Report No.: GR/R88137. |
| [83] | Gnu. Emacs. [Online].; 2014 [cited 2014 April 4]. Available from: <http://www.gnu.org/software/emacs>. |
| [84] | ErlIDE. erlide. [Online].; 2014 [cited 2014 April 4]. Available from: <http://erlide.org/>. |
| [85] | ignatov. github.io. [Online].; 2014 [cited 2014 April 4]. Available from: <http://ignatov.github.io/intellij-erlang>. |
| [86] | Waters K. allaboutagile.com. [Online].; 2009 [cited 2015 January 22]. Available from: <http://www.allaboutagile.com/prioritization-using-moscow/>. |
| [87] | coolfeature. GitHub. [Online].; 2015 [cited 2015 July 1]. Available from: <https://github.com/coolfeature/soil>. |
| [88] | coolfeature. GitHub. [Online].; 2015 [cited 2015 May 20]. Available from: <https://github.com/coolfeature/norm>. |
| [89] | ProcessOne. erlang-projects. [Online].; 2015 [cited 2015 June 15]. Available from: <http://tsung.erlang-projects.org/>. |
| [90] | dLuna. GitHub. [Online].; 2015 [cited 2015 June 15]. Available from: <https://github.com/dLuna/chaos_monkey>. |
| [91] | dLuna. README of ChaosMonkey. [Online].; 2015 [cited 2015 March 1]. Available from: <https://github.com/dLuna/chaos_monkey/blob/master/README>. |
| [92] | Hejlsberg A, Torgersen M. Overview of C# 3.0. [Online].; 2007 [cited 2015 April 1]. Available from: <https://msdn.microsoft.com/en-us/library/bb308966.aspx>. |
| [93] | Levick R. GitHub. [Online].; 2013 [cited 2015 June 15]. Available from: <http://rylev.github.io/words/blog/2013/10/03/erlang-is-the-most-object-oriented-language/>. |
| [94] | erlang.org. erlang.org. [Online].; 2015 [cited 2015 June 15]. Available from: <http://erlang.org/doc/man/reltool.html>. |
| [95] | rebar3.org. rebar3.org. [Online].; 2015 [cited 2015 June 17]. Available from: <https://www.rebar3.org/>. |
| [96] | erlang.org. erlang.org. [Online].; 2015 [cited 2015 June 17]. Available from: <http://www.erlang.org/doc/apps/eunit/chapter.html>. |
| [97] | propEr. propEr. [Online].; 2015 [cited 2015 June 17]. Available from: <http://proper.softlab.ntua.gr/doc/overview-summary.html>. |
| [98] | Boner J, Farley D, Kuhn R, Thompson M. The Reactive Manifesto. [Online].; 2015 [cited 2015 March 22]. Available from: <http://www.reactivemanifesto.org/>. |

x