UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

Delivering an Olympic Games

Autor: Agustín Treceño Orgaz

Tutor: Mario Muñoz Organero

This page intentionally left blank

Delivering an Olympic Games

Agustin Treceno

December 9, 2013

Dedicated to my parents.

Abstract

The technology involved in the distribution of the results during an Olympic Games is extremely complex. More than 900 servers, 1,000 network devices, 9,500 computers and 3,500 technologists are necessary to make it happen. Would it be possible to implement a solution with less resources using cutting edge technology? The following study answers this question by designing two scalable and high performance web-based applications to manage tournaments offering a REST interface to stakeholders. The first solution architecture is based on Java using frameworks such as Spring and Hibernate. The second solution architecture uses JavaScript with frameworks such as NodeJS, AngularJS, Mongoose and Express.

Contents

List of Figures

List of Tables

1 Introduction

People might doubt what you say.
But they will believe what you do.
—Lewis Cass

Chapter 1 gives an introduction to this study, including the reason behind it, the objectives and the structure of the document.

1.1 Motivation

The technology involved in the delivery of the results during an Olympic Games should not be taken for granted. Despite the fact that most of the software was reused from previous Olympic Games, there were more than 3,000 technologists who had been working for 4 years prior to the London 2012 opening ceremony. In a recent post by an editor of the New York Times, there was sufficient evidence to suggest companies struggled in setting up the environment to receive the Olympic information. What is it that makes the delivery of the results so difficult? Would it be possible to design a simpler solution by using new emerging frameworks and staying within budget?

1.2 Objectives

The purpose of this study is to design and implement a web based application to manage the sporting events of an Olympic Games. This implies that the application should be able to manage more than 300 sporting events, register more than 10,000 athletes, show the results of the competition and distribute those results to third party stakeholders.

1.3 About this document

1.3.1 Source code

This document was written entirely with the editor Vim in Markdown and compiled to Latex with Pandoc. In addition, some Agile software development methods, such as Scrum, Test-Driven Development (TDD) and Continuous Integration (CI), were used during this study. Agile methodology is a huge topic and it will not be covered here, however further reading can be found in Rubin (2012), Guckenheimer and Loje (2012) and Highsmith (2009).

All the source code of the applications along with this document can be found in my GitHub account[1].

1.3.2 Roadmap

Chapter 1 is a brief introduction to this study including the motivation behind this project and the main objectives.

Chapter 2 explains the current situation of the technology involved in the delivery of results during an Olympic Games.

Chapter 3 covers the first solution architecture, based on Java as a back-end. This solution uses frameworks such as Spring and Hibernate. Additionally, it shows how to use cloud computing to host the application in a production environment.

Chapter 4 discusses the second solution architecture, based on JavaScript as a back-end. Some of the technologies used in this solution are: Node.js, AngularJS, Express.js and Mongoose, to mention a few. The persistency is provided by a NoSQL database and the application is also hosted using cloud computing.

Chapter 5 includes some conclusions and future work.

[1] https://github.com/atreceno/

2 Behind the Olympics

There are two kinds of people, those who do the work and those who take the credit. Try to be in the first group; there is less competition there.
—Indira Gandhi

In this chapter, the state-of-the-art technology used in the distribution of results during an Olympic Games is reviewed. The London 2012 Olympic Games is the focus of this study. Section 2.1 presents an overview of the architecture and section 2.2 explains the format of the messages used.

2.1 Architecture

There is no doubt that the London 2012 Olympic Games have been the largest sporting event on the planet, to date (Rogge 2012). With over 10,000 athletes competing in 36 disciplines and over 300 medal events, the technology necessary to distribute the results in real-time to media and spectators all around the globe was extremely complicated. In the interests of confidentiality, this document does not provide a detailed outline of the architecture used at the 2012 Olympic Games. However, to give the reader an idea of how complex the solution was, here are some figures published by Atos, the worldwide IT partner of the Olympics:

- 2 million messages generated.
- 1,600 CIS[1] terminals.

[1] Commentator Information System (CIS) is a touch screen application for commentators.

- 1,800 Info[2] terminals.
- 3,500 technologists.
- 900 servers.
- 1,000 network and security devices.
- 9,500 computers.

The architectural overview of the results information system for the Olympic Games is shown in Figure 2.1:

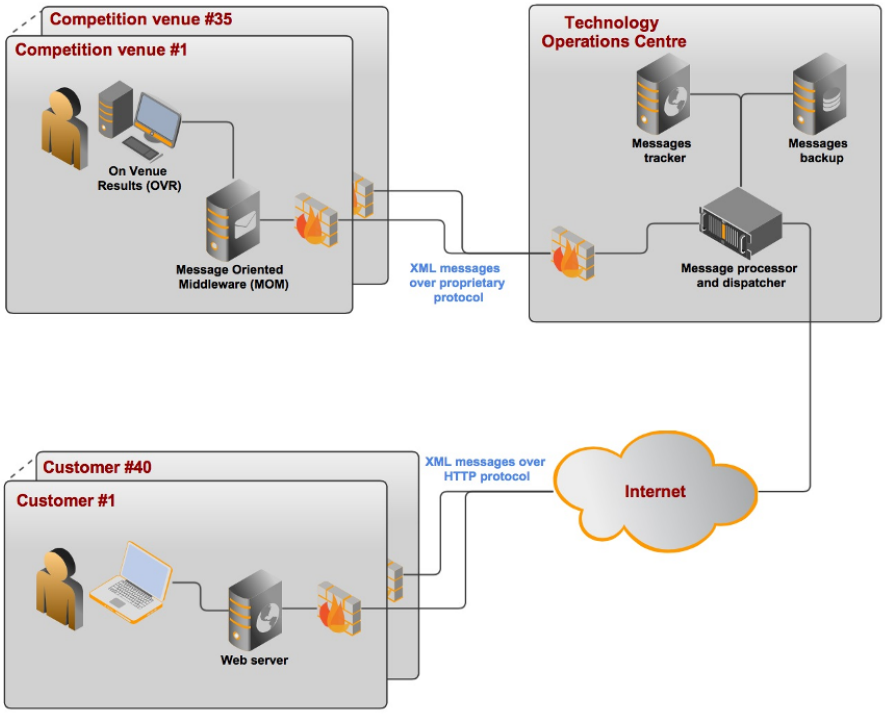

Figure 2.1: Architecture of the results information system

The information, such as results, schedules, start lists, medals, and so on, was encapsulated in XML messages. The structure of these messages has been defined by the International Olympic Committee (IOC) and is known by the name of Olympic Data Feed (ODF), which is briefly described in the next section.

Depending on the nature of the messages, they could be generated at the venues or centrally. For instance, start lists and results were generated at the venues, while biographies and news articles were generated from the central systems.

[2]Info is a web based application with information related to the Games such as schedules, results, news and weather conditions.

At the venue, messages were generated by Omega, the official timekeeper, from what is known as the On Venue Results (OVR). The results were generated automatically by transponders or other types of sensors; or manually triggered if the competition was not a race. These messages were then sent to the Message Oriented Middleware (MOM) at the venue and distributed to the corresponding queues. Once the messages were waiting at the venue, they were picked up from their queues by the *Message processor and dispatcher* using a proprietary protocol; then stored in the hard drive; and lastly, distributed to customers, such as sports federations and World News Press Agencies (WNPA). The XML messages were actually embedded into the body of an HTTP POST request issued by the dispatcher. This means, customers receiving this feed needed an application quite similar to a web server that was listening to HTTP requests sent by the dispatcher.

In addition to the *Commentator Information System* (CIS) and *Info* terminals, the results information system, a.k.a. Information Diffusion System (IDS), consisted of *Print Distribution* (PRD), an application to print reports automatically as soon as they were available; *Backup Internet Data Feed* (BIF), a web based application that hosted all messages that were sent; and several tools for managing and monitoring the system.

The Authentication, Authorisation and Accounting (AAA) was accomplished by a Lightweight Directory Access Protocol (LDAP), a database and a KEY server providing the passwords for both of them.

The distinct environments, including development, staging and production, were implemented at different levels: by using various physical hosts; logical hosts (using virtualisation); different ports in the same host; and at the application level by using, for example, different databases or queues.

The High Availability (HA) in the systems, not shown in the diagram, was provided by the Operating System (OS), in most cases. Both Solaris and Windows provided cluster services that were used during the disaster recovery tests.

2.2 The Olympic Data Feed

As mentioned in the previous section, the Olympic Data Feed (ODF) has been defined by the International Olympic Committee (IOC) in a set of documents available online[3]. The documents include the definition of attributes shared across all disciplines, as well as one data dictionary per discipline. In total, they sum 2630 pages just for the summer Olympic Games. The XML Schema Definition (XSD) is also available and describes the contract between the different parts of the system. It is possible to use the tool `xjc` to generate the Java classes out of the XSD file as follows:

[3] http://odf.olympictech.org/

```
$ xjc odf.xsd -p com.atreceno.it.odf -d src/main/java
```

The 256 Java classes generated can later be used by the Java Architecture for XML Binding (JAXB) library to map the XML messages into Java objects and viceversa.

Another interesting point is how the customers had to process each of those messages and keep the correct order, making the whole system *stateful*. For instance, a new score would trigger a message with the information of the competitor who scored, instead of the total result of the match. In this case the order may not be so important, however, if the message were for a participant update, the order would matter. To avoid this situation, two mechanisms are at play: a version field is included in all messages; and a cummulative result message is sent every so often.

This *stateful* behaviour made the whole system more complicated as each customer had to have their own persistence strategy to store the messages. Maher (2013), from *The New York Times*, explains how she had to deal with the Olympic data and how it took her team about 30,000 commits to the main branch to prepare for the Olympics. She also shows some of the ODF messages, reproduced here for our convenience:

```
<?xml version="1.0" encoding="utf-8"?>
<OdfBody DocumentCode="AT0000000"
  Serial="250428" Time="170759394" Date="20120712"
  FeedFlag="P" LogicalDate="20120712"
  DocumentType="DT_PARTIC" Version="51">
  <Competition Code="OG2012">
    <Participant Code="1083553"
      Parent="1083553" Status="ACCRED"
      GivenName="Dayron" FamilyName="Robles"
      PrintName="ROBLES Dayron"
      PrintInitialName="ROBLES D" TVName="Dayron ROBLES"
      TVInitialName="D. ROBLES" Gender="M"
      Organisation="CUB" BirthDate="19861119"
      PlaceofBirth="ISLA DE LA JUVENTUD"
      CountryofBirth="CUB" Nationality="CUB"
      MainFunctionId="AA01" Current="true"
      OlympicSolidarity="N">
      <Discipline Code="AT">
      </Discipline>
    </Participant>
    <Participant Code="1004617"
      Parent="1004617" Status="ACCRED"
      GivenName="Lyukman" FamilyName="Adams"
      PrintName="ADAMS Lyukman"
```

```
      PrintInitialName="ADAMS L" TVName="Lyukman ADAMS"
      TVInitialName="L. ADAMS" Gender="M"
      Organisation="RUS" BirthDate="19880924"
      Height="194" Weight="87"
      PlaceofBirth="LENINGRAD"
      CountryofBirth="RUS" Nationality="RUS"
      MainFunctionId="AA01" Current="true"
      OlympicSolidarity="N">
      <Discipline Code="AT"
        InternationalFederationId="208762">
        <RegisteredEvent
          Gender="M" Event="062">
          <EventEntry Code="E_PB"
            Type="E_ENTRY" Pos="1" Value="17.53" />
          <EventEntry Code="E_QUAL_BEST"
            Type="E_ENTRY" Pos="1" Value="17.53" />
          <EventEntry Code="E_SB"
            Type="E_ENTRY" Pos="1" Value="17.53" />
          <EventEntry Code="E_SUBSTITUTE"
            Type="E_ENTRY" Value="N" />
        </RegisteredEvent>
      </Discipline>
    </Participant>
    ...
  </Competition>
</OdfBody>
```

The above message is a fragment of the participant list for athletics during the London 2012 Olympics. A description of the main fields is shown below:

- `DocumentCode`: Alphanumeric code to identify the competition that the message refers to. It is composed of 9 characters `DDGEEEPUU`, where `DD` is the discipline code, `G` is the gender code, `EEE` is the event code, `P` is the phase code and `UU` is the event unit code.
- `DocumentType`: There are 88 different types of messages. For example, a `DT_SCHEDULE` contains information about when a competition takes place, `DT_RESULT` contains results, and so on.
- `Date` and `Time`: Date and time of when the message was generated.
- `LogicalDate`: Date of the competition. Typically, the date and *logical* date would be the same, except when the competition goes past midnight.
- `Version`: If the same code and type of message is sent, the version should be incremented.
- `Serial`: Similar to the version field but the serial number is reset to 0 at the beginning of the day.

- `FeedFlag`: It can have two values: `P` for production feed, and `T` for test feed. During a competition, only the production feed should be processed.

These fields are common to all ODF messages. Additionally, there is information specific to the type of message, in this case, information about the participants and the events they are registered in.

Below, a fragment of a `DT_RESULT` message is shown:

```
<OdfBody DocumentCode="ATM012101" DocumentType="DT_RESULT"
  FeedFlag="P" Date="20120808" Time="212121102"
  LogicalDate="20120808" Venue="STA" Version="1"
  ResultStatus="OFFICIAL" Serial="421">
  <Result ResultType="IRM" IRM="DQ" SortOrder="8">
    <Competitor Code="1083553" Type="A">
      <Composition>
        <Athlete Code="1083553" Order="1">
          <ExtendedResults>
            <ExtendedResult Code="AT_REACT_TIME"
              Type="UER_ATH_AT" Value="0.159" />
            <ExtendedResult Code="AT_RULE" Type="UER_ATH_AT"
              Value="R 168.7b" />
            <ExtendedResult Code="AT_WIND_SPEED"
              Type="UER_ATH_AT" Value="-0.3" />
          </ExtendedResults>
        </Athlete>
      </Composition>
    </Competitor>
  </Result>
  ...
</OdfBody>
```

The message corresponds to a result from Dayron Robles competing in the 110m hurdles event (`ATM012`) and shows an Invalid Result Mark (IRM). He was disqualified for violating the rule `R 168.7b`, which, according to the International Association of Athletics Federations (IAAF), means he deliberately knocked down a hurdle.

3

Spring Solution Architecture

The person who reads too much and uses his brain
too little will fall into lazy habits of thinking.
—Albert Einstain

Chapter 3 covers the first solution architecture, a web-based application using some well known software design patters such as, Dependency Injection (DI), Aspect-Oriented Programming (AOP) and Model-View-Controller (MVC). Additionally, the application, named *Diaulos*, was designed following good practices, e.g. loosely coupled components, separation of concerns, convention over configuration, to mention a few. Section 3.1 introduces SpringSource, the Inversion of Control (IoC) container used in this study. Section 3.2 covers the design and implementation from the ground up. It starts with the creation of the scaffolding with Spring Roo and continues with the persistence, service and web layers. Section 3.3 explains how to deploy the application using cloud computing. Finally, section 3.4 explores the limits of the application with some load tests.

3.1 Introduction to Spring

3.1.1 Features

Spring is an open source application framework, originally created by Rod Johnson, who released the framework with the publication of his book (Johnson 2003). The framework was created to address the complexity of enterprise applications by using a Plain Old Java Object (POJO) based programming model. Spring soon became an alternative to the more complex Enterprise JavaBean (EJB)

model (see Walls 2011, chap. 1; Ho and Harrop 2012, chap. 1–3). This simplification of the programming model, along with other features of the framework, are described below.

3.1.1.1 Based on Plain Old Java Object (POJO)

A Spring-based application should be made up of JavaBeans and should avoid, as much as possible, implementing a Spring-specific interface or extending a Spring-specific class. This way, the application is not littered with code from the framework and Java classes are just POJOs.

3.1.1.2 Loose coupling through Dependency Injection (DI) and interface orientation

Traditionally, in Object Oriented Programming (OOP) each object is responsible for obtaining its own references to the objects it collaborates with, i.e., its dependencies. This can, however, lead to highly coupled and hard-to-test code. With Inversion of Control (IoC), on the other hand, objects are given their dependencies at creation time by some third party that coordinates each object in the system. Hence the name of Inversion of Control (IoC), renamed by Martin Fowler to Dependency Injection (DI). DI is defined in the Java Specification Request JSR-330 and it is now part of the Java Enterprise Edition.

With DI, Java objects only know about their dependencies by their interfaces, as opposed to through their implementations. This way, the dependency can be swapped out with a different implementation (like those used by mocking frameworks) without the depending object knowing the difference. Prasanna (2009) explains DI in more detail.

3.1.1.3 Separation of concerns through Aspect-Oriented Programming (AOP)

Typically, an application is composed of several components, each responsible for a specific functionality. Sometimes all of these components make use of other concerns like logging, security or transaction management. These are usually referred to as cross-cutting concerns because they cut across multiple components in a program. With AOP it is possible to modularize those concerns and apply them declaratively, so the application is not littered with unnecessary code and POJOs remain plain. Laddad (2009) gives an excellent introduction to AOP and AspectJ.

3.1.1.4 Boilerplate reduction through templates

Boilerplate code is the code that has to be included in many places within an application, e.g. when querying data from a database using JDBC. Spring tries to eliminate boilerplate code by encapsulating it in templates. Another way of reducing boilerplate is by using the convention over configuration paradigm, which seeks to specify only unconventional aspects of the application by providing good default values.

3.1.2 Spring Modules

Spring Framework 3.2.4, released in August 2013, is made up of 19 different modules that can be grouped by functionality into 6 different categories (see Table 3.1). These categories are described below:

- **Core container**: Provides the fundamental parts of the framework such as the Inversion of Control (IoC) implementation, the bean factory, the application context and the expression language, which allows to manipulate Java objects at runtime.
- **Data access and integration**: Provides a JDBC-abstraction layer as well as support for Object Relational Mapping (ORM) implementations such as Java Persistence API (JPA), Java Data Objects (JDO), Hibernate and iBatis. Furthermore, it provides support for Object/XML mapping (OXM) implementations like Java Architecture for XML Binding (JAXB), Castor and XStream.
- **Web and remote**: Provides support for web applications including web application context, Spring's own MVC framework and portlets. It also provides remote access to other applications.
- **Aspect-oriented programming**: Provides an AOP implementation functionality and support for AspectJ.
- **Instrumentation**: Provides an instrumentation agent for JVM bootstrapping. It is required for using load-time weaving with AspectJ.
- **Testing**: Provides mock object implementations for writing unit tests against code, as well as support for integration testing.

Category	Module
Core container	spring-beans
Core container	spring-context
Core container	spring-context-support
Core container	spring-core

Core container	spring-expression
Data access and integration	spring-jdbc
Data access and integration	spring-jms
Data access and integration	spring-orm
Data access and integration	spring-oxm
Data access and integration	spring-tx
Web and remote	spring-struts
Web and remote	spring-web
Web and remote	spring-webmvc
Web and remote	spring-webmvc-portlet
Aspect-oriented programming	spring-aop
Aspect-oriented programming	spring-aspects
Instrumentation	spring-instrument
Instrumentation	spring-instrument-tomcat
Testing	spring-testing

Table 3.1: Spring Modules

Additionally, Spring has several projects built on top of the framework, such as Spring Batch, Spring Data, Spring Integration and Spring Roo. For a complete list of Spring projects, visit the Spring Projects website[1].

3.1.3 The IoC Container

In a Spring application, objects live within the Spring container, which is responsible for creating the objects, wiring them together, configuring them and managing their complete life cycle. These objects, a.k.a. beans, and their dependencies are defined in the configuration metadata through XML, Java annotations or Java code (see Walls 2011, chap. 2, 3; Ho and Harrop 2012, chap. 4, 5). Spring has several container implementations that can be grouped into 2 interfaces:

- `BeanFactory`: Provides basic support for dependency injection. Some implementing classes are `XMLBeanFactory`, `SimpleJndiBeanFactory` and `DefaultListableBeanFactory`.

[1] http://spring.io/projects

- `ApplicationContext`: Not only provides methods for accessing beans but also resolves messages with internationalisation, loads file resources, publishes events to registered listeners and inherits from a parent context. Some implementing classes are `FileSystemXMLApplicationContext` and `XmlWebApplicationContext`.

There are many ways in which beans can be wired together via XML. The following elements can be defined within the bean context definition:

- `constructor-arg` element allows for an argument to be passed to the bean constructor.
- `factory-method` element allows for a factory method to be used when the bean does not have a public constructor.
- `init-method` and `destroy-method` elements allow setup and teardown methods to be defined for a bean.
- `default-init-method` and `default-destroy-method` elements allow for setup and teardown methods to be defined across all beans in a given context definition.
- `property` element allows for the injection of properties by calling the respective setter. It is possible to pass single values, collections, references to other beans and even to use the Spring Expression Language (SpEL). Another way of wiring properties is by using Spring's `p` namespace.

By default, beans are `singleton`. Meaning the exact same instance of the bean will be dispensed each time it is asked for. To change this behaviour, the attribute `scope` is used when declaring a bean. The possible values are `singleton`, `prototype`, `request`, `session` and `global-session`.

Wiring beans together in an XML context gives great flexibility but sometimes, configuration can become a burden, especially in big applications. This is when *autowiring* and *autodiscovery* come into play. In Spring, *autowiring* is set up by using the `autowire` property in the bean definition, allowing for 4 different values:

- `byName`: Spring will attempt to match all bean properties with beans that have the same name.
- `byType`: In a similar way, bean properties will be matched with beans that have the same type. If there are several beans with the same type, it is possible to specify a primary candidate or even eliminate some beans from *autowiring*.
- `constructor`: Same as `byType` but using the arguments of the bean constructor instead of the bean properties.
- `autodetect`: Spring will try to wire the bean by `constructor` first, and then `byType`.

It is also possible to wire beans automatically through annotations, eliminating the use of `<constructor-arg>` and `<property>` elements. To use annotations, the element `<context:annotation-config />` should be present in the XML configuration file. Then, methods or properties in the class definition are annotated with `@Autowired`. Spring uses the `byType` *autowiring* and, to avoid ambiguity, it is possible to specify the name of the bean with the `@Qualifier` annotation. The Java Community Process (JCP) has defined two annotations in the JSR-330 specification, `@Inject` and `@Named`. They can be used instead of Spring's annotations `@Autowired` and `@Qualifier`, respectively.

To enable *autodiscovery* of beans, the `<context:annotation-config />` element should be replaced by `<context:component-scan />`. The container will then scan the package specified with the attribute `base-package` and look for classes annotated with `@Component` or any of its variants: `@Controller`, `@Service` or `@Repository`. Spring will create the corresponding beans by camelcasing the class name.

The element `<context:component-scan />` will also look for Java-based configuration classes, i.e. annotated with `@Configuration`. Beans are then defined by using the `@Bean` annotation (instead of the `<bean>` element in the XML configuration file) and wired by using constructors, setters or references to other beans.

3.1.4 Aspect-Oriented Programming (AOP)

In software engineering, the *divide and conquer* strategy can be one of the best approaches to tackling complexity. This is accomplished by breaking down the application into modules, each of which is responsible for solving a specific functionality or concern, leading to the well known practice separation of concerns. The Object-Oriented Programming (OOP) paradigm works quite well with core concerns by separating interfaces from their implementations. However, with cross-cutting concerns, this leads to code tangling[2] and code scattering[3]. With the Aspect-Oriented Programming (AOP) paradigm, common tasks are separated into *aspects*. An *aspect* is made up of an *advice*, which defines both the *what* and the *when*, and *pointcuts*, which define the *where*. In AOP terms, the job of an *aspect* is an *advice*, and it can be executed before or after a *join point*. A *join point* could be a method being called (this is the only one supported by Spring), an exception being thrown or even a field being modified. In short, *pointcuts* define which *join points* get advised (see Walls 2011, chap. 4; Ho and Harrop 2012, chap. 6, 7).

The process of applying aspects to a target object is called *weaving*, which can take place at compile time, *classload* time or run time. Spring Roo weaves aspects at compile time, while Spring AOP weaves them at run time by wrapping

[2]The code is mixed with code that implements other concerns.
[3]The code is spread out over multiple modules.

beans with a proxy class. This is why Spring AOP can be seen as an implementation of the decorator pattern, described in Gamma (1995). Other AOP frameworks include AspectJ and JBoss AOP, although JBoss AOP is no longer active.

3.1.5 Alternatives to Spring

Spring is not the only framework offering Dependency Injection features. Here are some alternatives to Spring:

- JBoss Seam Framework: Like Spring, it provides a fully-integrated development platform for building rich web applications in Java. It integrates technologies such as JavaServer Faces (JSF), Java Persistence API (JPA), Enterprise JavaBeans (EJB), Java Message Service (JMS) and even a module to connect Spring applications. The Java specification Context and Dependency Injection (CDI), defined in JSR-299, was a contribution from the Seam framework. The main difference with Spring is that JBoss Seam is built entirely on Java Enterprise Edition (JEE) standards. JBoss Seam 3 was released in January 2012 under the GNU Lesser General Public License (LGPL). The development of JBoss Seam 3 has been halted by Red Hat and many Seam projects have been moved to Apache DeltaSpike.
- Google Guice: Guice (pronounced *juice*) was born out of use cases from *AdWords*, one of Google's main advertising products. It provides Dependency Injection (DI) and uses annotations to create Java objects. Guice 3.0 was released in March 2011 under the Apache License 2.0.
- PicoContainer: PicoContainer is a lightweight Dependency Injection (DI) tool. It is used similar to a hash table, where `java.lang.Class` objects are added calling the `addComponent()` method, and object instances are returned calling the `getComponent()` method. The latest version is PicoContainer 2.10.2 released in February 2010, with a size of 308 KB.
- Silk DI: Silk DI is an even lighter Dependency Injection (DI) tool. The file size of the JAR file (only binaries) is only 188 KB. It is configurable via Java code instead of XML or annotations. The latest version of Silk is 0.6, released in July 2013.

Figure 3.1 shows how often these frameworks were searched on the Internet in the last 3 years. The values, expressed in percentage, are relative to the total search volume. Spring has been the most popular framework at 78.1%, followed by JBoss Seam (13.3%), Google Guice (5.7%), Pico Container (1.9%) and Silk DI (1%).

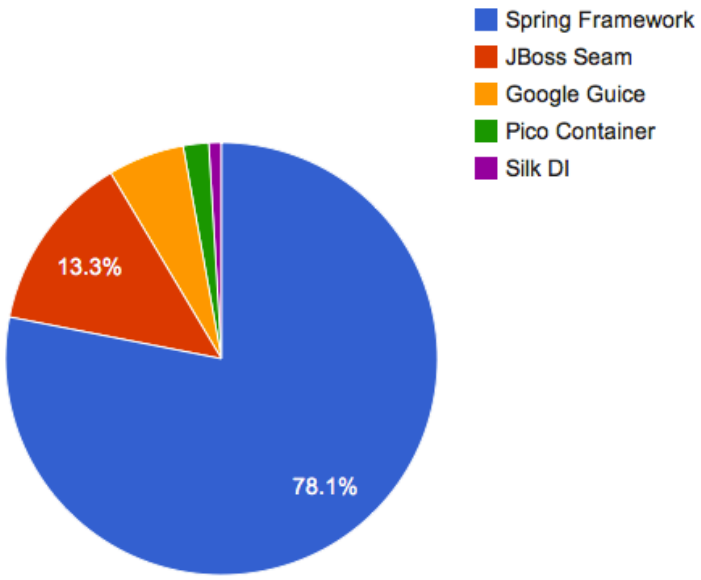

Figure 3.1: DI trends from July 2010 to July 2013 (source: Google Trends)

3.2 Design and Implementation

3.2.1 Java Scaffolding

Designing a modular enterprise Java application can be quite challenging nowadays, with so many options to consider: persistence, security, web framework, transaction support, validation process, to mention just a few. Thankfully, Spring Framework tries to address this problem by providing several modules, so developers can focus more on the business logic. To improve the productivity, Spring has a Rapid Application Development (RAD) tool called Spring Roo. Of course, Roo does not write any business logic, or add style to a website, but it is a good starting point to design a Spring application. Another advantage of using Roo is that it uses the convention over configuration principle, meaning that it tries to follow some convention or good practices unless otherwise specified. For example, given a class `EventGender` in the model, it creates a table `event_gender` in the database. A different name can be specified but, by default, it joins the words in lower-case through an underscore. Roo sticks to the same convention in the entire project. For those exceptions where a different behaviour is needed, it is possible to overwrite that unconventional aspect of the application by *pushing-in* the code or even adding new functionality by writing custom add-ons (see Rimple, Penchikala, and Alex 2012, chap. 1, 2).

Roo provides a command-line shell to add different components to the application. STS 3.4.0 comes with the latest version (1.2.4 at the time of writing) and it can be launched from STS itself (Ctrl-R on a Roo project) or from the command line.

A new project was created by issuing the command `project` as shown in the next listing:

```
$ roo
    ____  ____  ____
   / __ \/ __ \/ __ \
  / /_/ / / / / / / /
 / _, _/ /_/ / /_/ /
/_/ |_|\____/\____/    1.2.4.RELEASE [rev 75337cf]


Welcome to Spring Roo. For assistance press TAB or type "hint"
then hit ENTER.
roo>
roo> project --topLevelPackage com.atreceno.it.diaulos
--projectName diaulos --java 7 --packaging JAR
Created ROOT/pom.xml
Created SRC_MAIN_RESOURCES
Created SRC_MAIN_RESOURCES/log4j.properties
```

```
Created SPRING_CONFIG_ROOT
Created SPRING_CONFIG_ROOT/applicationContext.xml
roo> exit
```

Roo created the following files:

- `log.roo`: Maintains a list of commands that have been used.
- `pom.xml`: Is the Maven project descriptor or Project Object Model.
- `src/main/resources/log4j.properties`: Contains the configuration for the logging library Apache Log4j.
- `src/main/resources/META-INF/spring/applicationContext.xml`: Contains the configuration for Spring.

To keep track of Roo's changes, it is highly recommended to use a version control system. Therefore, the next step was to create a Git repository in the project's folder:

```
$ git init
Initialized empty Git repository in
/Users/agustin/Development/git/diaulos/.git/
```

With the following `.gitignore` file:

```
$ vi .gitignore
$ cat $_
# Mac OS X
.DS_Store
# Eclipse IDE
/target
/.project
/.classpath
/.settings
# Spring Roo
/log.roo
# Credentials
/src/main/resources/META-INF/spring/database.properties
/src/main/resources/META-INF/sql/user.xml
```

The files `database.properties` and `user.xml` were added to `.gitignore` because they contained information on credentials to access the database and the application, respectively. They were replaced by templates so the files could be part of the version control system but the credentials were only in the application server.

Once the Git repository was initialised, the code was added as follows:

```
$ git add .
$ git ls
.gitignore
pom.xml
src/main/resources/META-INF/spring/applicationContext.xml
src/main/resources/log4j.properties
$ git commit
[master (root-commit) adbf9e0] Create scaffolding with Roo
 4 files changed, 373 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 pom.xml
 create mode 100644 src/resources/.../applicationContext.xml
 create mode 100644 src/main/resources/log4j.properties
$ git branch jpa
$ git branch mongo
$ git branch
  jpa
* master
  mongo
```

Next, a remote repository was defined:

```
$ pwd
/Users/agustin/Development/git
$ git clone --bare diaulos diaulos.git
Cloning into bare repository 'diaulos.git'...
done.
$ scp -r diaulos.git \
agustin@sochi.gast.it.uc3m.es:~/Development/git
$ cd diaulos
$ git remote add uc3m \
agustin@sochi.gast.it.uc3m.es:~/Development/git/diaulos.git
$ git push uc3m
Everything up-to-date
$ rm -rf ../diaulos.git
```

To this point, the basic structure of the project was defined and it could be imported into STS from the local repository by selecting *Existing Maven Projects* from the *Import* menu.

3.2.2 Persistence Layer

3.2.2.1 The Model

There are so many alternatives when it comes to defining the model of an application: database normalization, database relationships, natural keys and surrogate keys, are some of them. The diverse nature of the different types of competitions does not make things easier. Furthermore, trying to model different sporting events under the same pattern can become quite a challenge, especially, with relational databases. Hernandez (2003) and Garcia-Molina, Ullman, and Widom (2008) are both good references to understand how relational databases work. Figure 3.2 shows the Enhanced Entity-Relationship (EER) model of a tournament management tool that could be used during an Olympic Games. In this case it shows only races because this is the type of competition that generates the most traffic, but it could be extrapolated to any other type of competition.

There is one main branch: sport > event > phase > race > lap. Meaning *sports* have *events*, *events* have *phases*, and so on. Then a *participant* table is linked in a Many-to-many relationship with the previous tables. This not only allows *participants* to be registered in *events*, but also to compete in *races* and have a ranking at *phase* level. The *user* and *role* tables allow users to access the application according to the defined roles.

3.2.2.2 Configuring JPA

Spring Roo was used to configure a Java Persistence API (JPA) persistence layer. JPA is a specification written by the Java Community Process (JCP) group and it describes the management of relational data in a Java application. JPA 2.1 has been recently approved as final in JSR-338 but Roo uses JPA 2.0, defined under the Java Specification Request JSR-317:

> The Java Persistence 2.0 specification addresses improvements in the areas of domain modeling, object/relational mapping, EntityManager and Query interfaces, and the Java Persistence query language. It adds an API for criteria queries, a metamodel API, and support for validation.

The Roo command used to setup a JPA persistence provider is `jpa setup` (see Rimple, Penchikala, and Alex 2012, chap. 3, 4). Roo supports several providers, i.e. JPA implementations: Hibernate, EclipseLink, OpenJPA. Roo equally supports several databases: DB2, Oracle, MySQL and Hypersonic, to mention a few. In this study, Hibernate was used as a JPA implementation and MySQL as a database. Both are very popular in the Java community and work quite well with Spring (see Fisher and Murphy 2010). In the following listing the command `jpa setup` is issued specifying a database name and username:

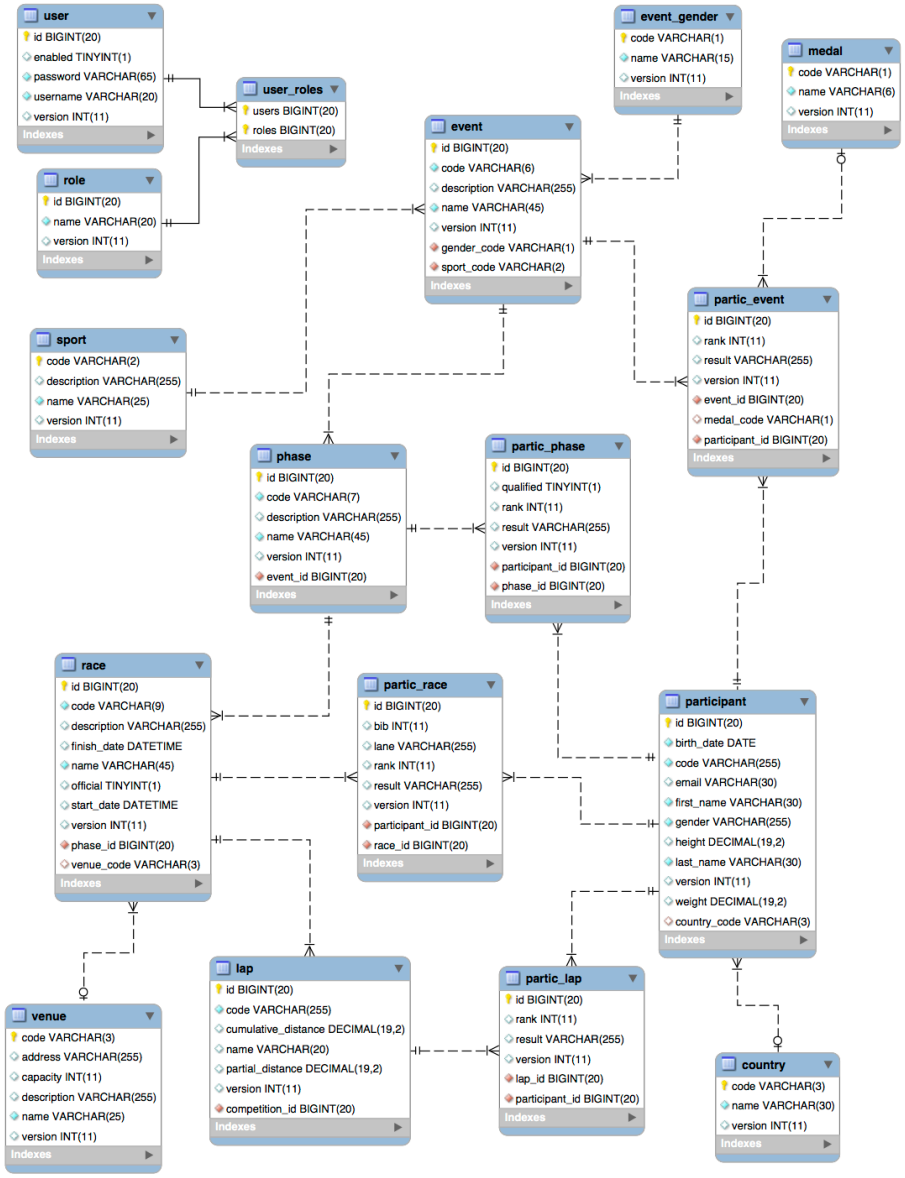

Figure 3.2: EER Model of a sporting event (Crow's foot notation)

```
$ git checkout jpa
Switched to branch 'jpa'
$ git branch
* jpa
  master
  mongo
$ roo
roo> jpa setup --provider HIBERNATE --database MYSQL
--databaseName diaulos_db --userName bricklane
Created SPRING_CONFIG_ROOT/database.properties
Updated SPRING_CONFIG_ROOT/applicationContext.xml
Created SRC_MAIN_RESOURCES/META-INF/persistence.xml
Updated ROOT/pom.xml [added dependencies
mysql:mysql-connector-java:5.1.18,
org.hibernate:hibernate-core:4.1.8.Final,
org.hibernate:hibernate-entitymanager:4.1.8.Final,
org.hibernate.javax.persistence:hibernate-jpa-2.0-api:1.0.1.Final,
commons-collections:commons-collections:3.2.1,
org.hibernate:hibernate-validator:4.3.1.Final,
javax.validation:validation-api:1.0.0.GA,
cglib:cglib-nodep:2.2.2,
javax.transaction:jta:1.1,
org.springframework:spring-jdbc:${spring.version},
org.springframework:spring-orm:${spring.version},
commons-pool:commons-pool:1.5.6,
commons-dbcp:commons-dbcp:1.3]
roo> quit
$ git add .
$ git status
# On branch jpa
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   pom.xml
#   new file:   src/main/resources/META-INF/persistence.xml
#   modified:   src/.../META-INF/spring/applicationContext.xml
#   new file:   src/.../spring/database.template.properties
#
$ git commit -m "Add JPA provider Hibernate and database MySQL"
[jpa 57b6547] Add JPA provider Hibernate and database MySQL
 4 files changed, 128 insertions(+), 4 deletions(-)
 create mode 100644 src/main/resources/META-INF/persistence.xml
 create mode 100644 src/.../spring/database.template.properties
```

For the sake of brevity, only the relevant Roo commands will be shown. For the complete list of commands, the reader can refer to the script `diaulos.roo`.

Notice how Roo added several dependencies to the Maven configuration file `pom.xml`: the MySQL connector; several Hibernate libraries; Java Transaction API (JTA) defined in the JSR-907 specification; Bean Validation API defined in the JSR-303 specification; and 2 Spring modules: `spring-jdbc` and `spring-orm`.

Additionally, the root application context was updated with three new components: a JDBC datasource, a transaction manager and an entity manager factory. The file `META-INF/spring/applicationContext.xml` with the mentioned components is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<beans
  xmlns="http://www.sf.org/schema/beans"
  xmlns:aop="http://www.sf.org/schema/aop"
  xmlns:context="http://www.sf.org/schema/context"
  xmlns:jee="http://www.sf.org/schema/jee"
  xmlns:tx="http://www.sf.org/schema/tx"
  xmlns:jdbc="http://www.sf.org/schema/jdbc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.sf.org/schema/aop
    http://www.sf.org/schema/aop/spring-aop-3.2.xsd
    http://www.sf.org/schema/beans
    http://www.sf.org/schema/beans/spring-beans-3.2.xsd
    http://www.sf.org/schema/context
    http://www.sf.org/schema/context/spring-context-3.2.xsd
    http://www.sf.org/schema/jee
    http://www.sf.org/schema/jee/spring-jee-3.2.xsd
    http://www.sf.org/schema/tx
    http://www.sf.org/schema/tx/spring-tx-3.2.xsd
    http://www.sf.org/schema/jdbc
    http://www.sf.org/schema/jdbc/spring-jdbc-3.2.xsd">

  <context:property-placeholder
    location="classpath*:META-INF/spring/*.properties" />
  <context:spring-configured />
  <context:component-scan base-package="com.atreceno.it.diaulos">
    <context:exclude-filter
      expression=".*_Roo_.*" type="regex" />
    <context:exclude-filter
      expression="org.sf.stereotype.Controller"
      type="annotation" />
  </context:component-scan>
  <bean class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close" id="dataSource">
    <property name="driverClassName"
```

```
      value="${database.driverClassName}" />
    <property name="url" value="${database.url}" />
    <property name="username" value="${database.username}" />
    <property name="password" value="${database.password}" />
    <property name="testOnBorrow" value="true" />
    <property name="testOnReturn" value="true" />
    <property name="testWhileIdle" value="true" />
    <property name="timeBetweenEvictionRunsMillis"
      value="1800000" />
    <property name="numTestsPerEvictionRun" value="3" />
    <property name="minEvictableIdleTimeMillis"
      value="1800000" />
    <property name="validationQuery" value="SELECT 1" />
  </bean>
  <bean class="org.sf.orm.jpa.JpaTransactionManager"
    id="transactionManager">
    <property name="entityManagerFactory"
      ref="entityManagerFactory" />
  </bean>
  <tx:annotation-driven
    mode="aspectj" transaction-manager="transactionManager" />
  <bean
    class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean"
    id="entityManagerFactory">
    <property name="persistenceUnitName"
      value="persistenceUnit" />
    <property name="dataSource" ref="dataSource" />
  </bean>
  <jdbc:initialize-database>
    <jdbc:script
      location="classpath:/META-INF/sql/diaulos_dml.sql" />
  </jdbc:initialize-database>
</beans>
```

Two other files were created as a result of the command `jpa setup`:

- `META-INF/spring/database.properties` contains information on credentials to access the database, as mentioned in the previous section.
- `META-INF/persistence.xml` is the standard JPA configuration file and contains some configuration parameters for the selected Object-Relational Mapping (ORM) library, i.e. Hibernate (see Linwood and Minter 2010).

The file `META-INF/persistence.xml` is shown in the next listing:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<persistence
```

```
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="2.0"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="persistenceUnit"
    transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.MySQL5InnoDBDialect" />
      <property name="hibernate.hbm2ddl.auto" value="create" />
      <property name="hibernate.ejb.naming_strategy"
        value="org.hibernate.cfg.ImprovedNamingStrategy" />
      <property name="hibernate.connection.charSet"
        value="UTF-8" />
      <property name="hibernate.show_sql" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

For more information about Spring's data access support, Walls (2011, chap. 5) and Ho and Harrop (2012, chap. 8–11) are excellent references.

3.2.2.3 Creating Entities

The next step in the definition of the persistence layer is to create some JPA entities. A JPA entity is a class that is mapped to a table in the database. Roo has two ways of creating a JPA entity: The more traditional approach using a *repository* layer; or using the Active record pattern which is more popular among Rapid Application Development (RAD) frameworks like Ruby on Rails or Grails. Both alternatives are equally valid but Roo only provides finders for the latter. In an Active record pattern, entities contain everything they need to update and persist themselves. Fowler (2003, 160–162) was the first one introducing this pattern. Below, are two examples of entities: `Sport`, with `--activeRecord` parameter set to `true`, and `Event` with `--activeRecord` parameter set to `false`:

```
roo> entity jpa --class ~.domain.Sport --equals
--activeRecord true --testAutomatically --identifierField code
--identifierType java.lang.String
Created SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain
Created SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain/Sport.java
Created SRC_TEST_JAVA/com/atreceno/it/diaulos/domain
Created SRC_TEST_JAVA/.../SportDataOnDemand.java
Created SRC_TEST_JAVA/.../SportIntegrationTest.java
```

```
Created SRC_MAIN_JAVA/.../Sport_Roo_Configurable.aj
Created SRC_MAIN_JAVA/.../Sport_Roo_ToString.aj
Created SRC_MAIN_JAVA/.../Sport_Roo_Jpa_Entity.aj
Created SRC_MAIN_JAVA/.../Sport_Roo_Jpa_ActiveRecord.aj
Created SRC_MAIN_JAVA/.../Sport_Roo_Equals.aj
Created SRC_TEST_JAVA/.../SportIntegrationTest_Roo_Configurable.aj
Created SRC_TEST_JAVA/.../SportDataOnDemand_Roo_DataOnDemand.aj
Created SRC_TEST_JAVA/.../SportIntegrationTest_Roo_IntegrationTest.aj
Created SRC_TEST_JAVA/.../SportDataOnDemand_Roo_Configurable.aj
~.domain.Sport roo>
~.domain.Sport roo> entity jpa --class ~.domain.Event --equals
--activeRecord false --testAutomatically
Created SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain/Event.java
Created SRC_TEST_JAVA/.../EventDataOnDemand.java
Created SRC_TEST_JAVA/.../EventIntegrationTest.java
Created SRC_MAIN_JAVA/.../Event_Roo_ToString.aj
Created SRC_MAIN_JAVA/.../Event_Roo_Jpa_Entity.aj
Created SRC_MAIN_JAVA/.../Event_Roo_Equals.aj
Created SRC_TEST_JAVA/.../EventDataOnDemand_Roo_Configurable.aj
Created SRC_TEST_JAVA/.../EventIntegrationTest_Roo_Configurable.aj
~.domain.Event roo>
```

The parameter `--testAutomatically` instructs Roo to create the test classes `{Entity}DataOnDemand` and `{Entity}IntegrationTest`, which were used to generate the necessary test data and include the integration tests, respectively.

The files with the extension `.aj` are AspectJ ITD (inter-type declaration) managed by Roo that are woven into the corresponding classes at compile time using the Maven plugin aspectj-maven-plugin. They contain java code but are defined by the keyword `aspect` instead of `class`. It is also possible to extract methods from these files to customise them (process known as *push-in* refactoring), or even remove Roo completely from the application.

The command `entity jpa` created empty domain classes. To add properties to those JavaBeans, the command `field` was used as follows:

```
~.domain.Event roo> focus --class ~.domain.Sport
~.domain.Sport roo> field string --fieldName code --notNull
--sizeMin 2 --sizeMax 2
Updated SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain/Sport.java
Updated SRC_TEST_JAVA/.../SportDataOnDemand_Roo_DataOnDemand.aj
Updated SRC_TEST_JAVA/.../SportIntegrationTest_Roo_IntegrationTest.aj
Updated SRC_MAIN_JAVA/.../Sport_Roo_Equals.aj
Updated SRC_MAIN_JAVA/.../Sport_Roo_Jpa_Entity.aj
Updated SRC_MAIN_JAVA/.../Sport_Roo_Jpa_ActiveRecord.aj
Created SRC_MAIN_JAVA/.../Sport_Roo_JavaBean.aj
```

```
~.domain.Sport roo> field string --fieldName name --notNull
--sizeMax 25
Updated SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain/Sport.java
Updated SRC_MAIN_JAVA/.../Sport_Roo_Equals.aj
Updated SRC_TEST_JAVA/.../SportDataOnDemand_Roo_DataOnDemand.aj
Updated SRC_MAIN_JAVA/.../Sport_Roo_JavaBean.aj
~.domain.Sport roo> field set --fieldName events
--type ~.domain.Event --mappedBy sport --cardinality ONE_TO_MANY
Updated SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain/Sport.java
Updated SRC_MAIN_JAVA/.../Sport_Roo_JavaBean.aj
~.domain.Sport roo>
~.domain.Sport roo> focus --class ~.domain.Event
~.domain.Event roo> field string --fieldName code --notNull
--sizeMin 6 --sizeMax 6 --unique
Updated SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain/Event.java
Updated SRC_MAIN_JAVA/.../domain/Event_Roo_Equals.aj
Created SRC_MAIN_JAVA/.../domain/Event_Roo_JavaBean.aj
~.domain.Event roo> field string --fieldName name --notNull
--sizeMax 45
Updated SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain/Event.java
Updated SRC_MAIN_JAVA/.../domain/Event_Roo_Equals.aj
Updated SRC_MAIN_JAVA/.../domain/Event_Roo_JavaBean.aj
~.domain.Event roo> field reference --fieldName sport
--type ~.domain.Sport --notNull --joinColumnName sport_code
Updated SRC_MAIN_JAVA/com/atreceno/it/diaulos/domain/Event.java
Updated SRC_MAIN_JAVA/.../domain/Event_Roo_Equals.aj
Updated SRC_MAIN_JAVA/.../domain/Event_Roo_JavaBean.aj
~.domain.Event roo>
```

AspectJ ITD files were actually created by the Roo shell in response to the annotations present in the class definition. By using AspectJ ITDs, there is a separation of concerns and the entities are clearer. The following listing shows the entity `Sport.java` with its corresponding properties:

```
package com.atreceno.it.diaulos.domain;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
```

```
import org.sf.roo.addon.equals.RooEquals;
import org.sf.roo.addon.javabean.RooJavaBean;
import org.sf.roo.addon.jpa.activerecord.RooJpaActiveRecord;
import org.sf.roo.addon.tostring.RooToString;

@RooJavaBean
@RooToString
@RooEquals
@RooJpaActiveRecord
public class Sport {

    @Id
    @NotNull
    @Size(min = 2, max = 2)
    private String code;

    @NotNull
    @Size(max = 25)
    private String name;

    @Size(max = 255)
    private String description;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "sport")
    private Set<Event> events = new HashSet<Event>();
}
```

Each annotation of the `Sport.java` class definition provided a different functionality:

- `@RooJavaBean`: JavaBean support.
- `@RooToString`: String representation of the entity.
- `@RooEquals`: Comparison between two entities.
- `@NotNull` and `@Size`: Bean validations.
- `@OneToMany`: Define entity relationships.
- `@RooJpaActiveRecord` or `@RooJpaEntity`: Active record pattern or JPA entity.

A more detailed explanation of each functionality can be found in the following sections.

3.2.2.4 JavaBean Support

When an entity is annotated with `@RooJavaBean`, its *getters* and *setters* are created in the aspect `{Entity}_Roo_JavaBean.aj`. Below is the source code

for Sport_Roo_JavaBean.aj:

```
package com.atreceno.it.diaulos.domain;

import com.atreceno.it.diaulos.domain.Event;
import com.atreceno.it.diaulos.domain.Sport;
import java.util.Set;

privileged aspect Sport_Roo_JavaBean {

    public String Sport.getCode() {
        return this.code;
    }

    public void Sport.setCode(String code) {
        this.code = code;
    }

    public String Sport.getName() {
        return this.name;
    }

    public void Sport.setName(String name) {
        this.name = name;
    }

    public String Sport.getDescription() {
        return this.description;
    }

    public void Sport.setDescription(String description) {
        this.description = description;
    }

    public Set<Event> Sport.getEvents() {
        return this.events;
    }

    public void Sport.setEvents(Set<Event> events) {
        this.events = events;
    }

}
```

3.2.2.5 Bean Validation

The `code` attribute in the `Sport.java` class was annotated with `@NotNull` and `@Size` annotations as a result of the command `field string --fieldName code --notNull --sizeMin 2 --sizeMax 2`. Roo is using the Bean Validation API JSR-303, which uses Java annotations attached to attributes to define specific rules. The API describes several constraints like `@NotNull`, `@Size`, `@Min`, `@Max`, `@Future`, `@Past`, and so on. It is also possible to overwrite or extend the framework and add custom validations. The library `hibernate-validator`, included in the project descriptor `pom.xml`, provides the reference implementation of Bean Validation.

3.2.2.6 Using Reflection

Roo uses the class `ReflectionToStringBuilder` from Apache `commons-lang3` library to create a string, out of the entity fields. It uses reflection to determine the fields to append. Following the source code of the aspect `Sport_Roo_toString.aj`:

```
package com.atreceno.it.diaulos.domain;

import com.atreceno.it.diaulos.domain.Sport;
import org.apache.commons.lang3.builder.ReflectionToStringBuilder;
import org.apache.commons.lang3.builder.ToStringStyle;

privileged aspect Sport_Roo_ToString {

    public String Sport.toString() {
        return ReflectionToStringBuilder
            .toString(this, ToStringStyle.SHORT_PREFIX_STYLE);
    }

}
```

The second argument of the method `toString()` allows to specify a format to represent the entity.

3.2.2.7 Comparing Entities

Roo also uses two other classes: `EqualsBuilder` and `HashCodeBuilder` from Apache `commons-lang3` library to implement the `equals()` and `hashCode()` methods. These two methods were located in the aspect `Sport_Roo_Equals.aj` as a result of the `@RooEquals` annotation:

```
package com.atreceno.it.diaulos.domain;

import com.atreceno.it.diaulos.domain.Race;
import org.apache.commons.lang3.builder.EqualsBuilder;
import org.apache.commons.lang3.builder.HashCodeBuilder;

privileged aspect Race_Roo_Equals {

    public boolean Race.equals(Object obj) {
        if (!(obj instanceof Race)) {
            return false;
        }
        if (this == obj) {
            return true;
        }
        Race rhs = (Race) obj;
        return new EqualsBuilder().append(code, rhs.code)
            .append(description, rhs.description)
            .append(finishDate, rhs.finishDate)
            .append(id, rhs.id)
            .append(name, rhs.name)
            .append(official, rhs.official)
            .append(phase, rhs.phase)
            .append(startDate, rhs.startDate)
            .append(venue, rhs.venue).isEquals();
    }

    public int Race.hashCode() {
        return new HashCodeBuilder().append(code)
            .append(description)
            .append(finishDate)
            .append(id)
            .append(name)
            .append(official)
            .append(phase)
            .append(startDate)
            .append(venue).toHashCode();
    }

}
```

3.2.2.8 Defining Entity Relationships

To relate JPA entities to each other, Roo provides two variants of the `field` command: `field reference` and `field set`. The first, maps a single reference,

the second, a collection of elements of a particular type. This way it is possible to create any JPA relationships (One-to-one, One-to-many, Many-to-one and Many-to-many) as a combination of these two variants.

For example, the relationship between a `Sport` and an `Event` was One-to-many so the `Sport` class was created as follows:

```
roo> focus --class ~.domain.Sport
roo> field set --fieldName events --type ~.domain.Event
--mappedBy sport --cardinality ONE_TO_MANY
```

The `Event` entity contained a reference to a `Sport`, a reference to a `Gender`, a collection of `Phases` and a collection of `Participants`, as shown in the following listing:

```
roo> focus --class ~.domain.Event
roo> field reference --fieldName gender
--type ~.domain.EventGender --notNull --joinColumnName gender_code
roo> field reference --fieldName sport --type ~.domain.Sport
--notNull --joinColumnName sport_code
roo> field set --fieldName phases --type ~.domain.Phase
--mappedBy event --cardinality ONE_TO_MANY
roo> field set --fieldName participants
--type ~.domain.ParticEvent --mappedBy event
--cardinality ONE_TO_MANY
```

Note how the participant's type is `ParticEvent` because there were actually two relationships (One-to-many and Many-to-one) instead of one (Many-to-many). It was also possible to define a Many-to-many relationship without using an intermediate entity, as shown below:

```
roo> focus --class ~.domain.security.User
roo> field string --fieldName username --notNull --sizeMin 3
--sizeMax 20
roo> field string --fieldName password --notNull --sizeMin 3
--sizeMax 65
roo> field boolean --fieldName enabled
roo> field set --fieldName roles --type ~.domain.security.Role
--cardinality MANY_TO_MANY
roo> focus --class ~.domain.security.Role
roo> field string --fieldName name --notNull --sizeMin 8
--sizeMax 20 --regexp ^ROLE_[A-Z]*
roo> field set --fieldName users --type ~.domain.security.User
--mappedBy roles --cardinality MANY_TO_MANY
```

3.2.2.9 Active Record Pattern

Roo annotates a class definition with `@RooJpaActiveRecord` when a JPA entity is created with the option `--activeRecord true`. In response to this, Roo generates three aspects:

- `{Entity}_Roo_Configurable.aj`
- `{Entity}_Roo_Jpa_Entity.aj`
- `{Entity}_Roo_Jpa_ActiveRecord.aj`

This is how `Country`, `EventGender`, `Medal` and `Sport` entities were created. For instance, the three aspects for the `Sport` entity were:

- `Sport_Roo_Configurable.aj`
- `Sport_Roo_Jpa_Entity.aj`
- `Sport_Roo_Jpa_ActiveRecord.aj`

The first aspect, `Sport_Roo_Configurable.aj`, is shown in the following listing:

```
package com.atreceno.it.diaulos.domain;

import com.atreceno.it.diaulos.domain.Sport;
import org.sf.beans.factory.annotation.Configurable;

privileged aspect Sport_Roo_Configurable {

    declare @type: Sport: @Configurable;

}
```

The Spring container creates and configures beans defined in the application context. It is equally possible to create and configure beans that are outside the scope of the container, i.e. created with the *new* operator or by an ORM framework as a result of a database query. To enable this, the tag `<context:spring-configured/>` has to be present in the application context and the Java class definition annotated with `@Configurable`. See Using AspectJ with Spring applications in the reference manual online[4] for more information.

The second aspect created as a result of the `@RooJpaActiveRecord` annotation was `Sport_Roo_Jpa_Entity.aj` containing the primary key and the version strategy:

[4] http://docs.spring.io/spring/docs/3.2.4.release/spring-framework-reference/html/

```
package com.atreceno.it.diaulos.domain;

import com.atreceno.it.diaulos.domain.Sport;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Version;

privileged aspect Sport_Roo_Jpa_Entity {

    declare @type: Sport: @Entity;

    @Version
    @Column(name = "version")
    private Integer Sport.version;

    public Integer Sport.getVersion() {
        return this.version;
    }

    public void Sport.setVersion(Integer version) {
        this.version = version;
    }

}
```

Finally, the third aspect, `Sport_Roo_Jpa_ActiveRecord.aj`, contained all JPA methods to perform CRUD (create, read, update and delete) operations to a `Sport` entity instance:

```
package com.atreceno.it.diaulos.domain;

import com.atreceno.it.diaulos.domain.Sport;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import org.sf.transaction.annotation.Transactional;

privileged aspect Sport_Roo_Jpa_ActiveRecord {

    @PersistenceContext
    transient EntityManager Sport.entityManager;

    public static final EntityManager Sport.entityManager() {
        EntityManager em = new Sport().entityManager;
        if (em == null) throw new IllegalStateException("Entity
```

```
        manager has not been injected (is the Spring Aspects
        JAR configured as an AJC/AJDT aspects library?)");
    return em;
}

public static long Sport.countSports() {
    return entityManager().createQuery(
        "SELECT COUNT(o) FROM Sport o",
        Long.class).getSingleResult();
}

public static List<Sport> Sport.findAllSports() {
    return entityManager().createQuery(
        "SELECT o FROM Sport o",
        Sport.class).getResultList();
}

public static Sport Sport.findSport(String code) {
    if (code == null || code.length() == 0) return null;
    return entityManager().find(Sport.class, code);
}

public static List<Sport> Sport
    .findSportEntries(int firstResult, int maxResults) {
        return entityManager().createQuery(
            "SELECT o FROM Sport o", Sport.class)
            .setFirstResult(firstResult)
            .setMaxResults(maxResults).getResultList();
}

@Transactional
public void Sport.persist() {
    if (this.entityManager == null) {
        this.entityManager = entityManager();
    }
    this.entityManager.persist(this);
}

@Transactional
public void Sport.remove() {
    if (this.entityManager == null) {
        this.entityManager = entityManager();
    }
    if (this.entityManager.contains(this)) {
        this.entityManager.remove(this);
    } else {
```

```
            Sport attached = Sport.findSport(this.code);
            this.entityManager.remove(attached);
        }
    }

    @Transactional
    public void Sport.flush() {
        if (this.entityManager == null) {
            this.entityManager = entityManager();
        }
        this.entityManager.flush();
    }

    @Transactional
    public void Sport.clear() {
        if (this.entityManager == null) {
            this.entityManager = entityManager();
        }
        this.entityManager.clear();
    }

    @Transactional
    public Sport Sport.merge() {
        if (this.entityManager == null) {
            this.entityManager = entityManager();
        }
        Sport merged = this.entityManager.merge(this);
        this.entityManager.flush();
        return merged;
    }

}
```

The static method `entityManager()` creates a new `Sport` entity and returns the injected JPA entity manager, i.e. `entityManager` instance. This is possible because the class definition of the entity `Sport` was annotated with `@Configurable`.

There are two ways of doing declarative transaction management in Spring: via XML or annotations. In this case, the latter was chosen and, consequently, the tag `<tx:annotation-driven />` was added to the application context and the methods annotated with `@Transactional`.

3.2.2.10 Repository Pattern

As it has already been mentioned, Roo provides two ways of accessing the model via JPA: Active record pattern, and the more traditional *repository* layer. The `Event` entity was created using the latter. The class definition of the entity had an `@RooJpaEntity` annotation instead of `@RooJpaActiveRecord`. As a result the Roo shell created the aspect `Event_Roo_Jpa_Entity.aj` containing the ID and version fields and the corresponding *getters* and *setters*.

To create a repository that would back the `Event` entity, the command `repository jpa` was used:

```
roo> repository jpa --interface ~.repository.EventRepository
--entity ~.domain.Event
Created SRC_MAIN_JAVA/com/atreceno/it/diaulos/repository
Created SRC_MAIN_JAVA/com/.../repository/EventRepository.java
Created SPRING_CONFIG_ROOT/applicationContext-jpa.xml
Updated ROOT/pom.xml [added dependency org.sf.data:
spring-data-jpa:1.2.0.RELEASE]
Created SRC_MAIN_JAVA/.../EventRepository_Roo_Jpa_Repository.aj
```

The library `spring-data-jpa` was added to the project descriptor `pom.xml`. The file `META-INF/spring/applicationContext-jpa.xml` that was created is a JPA-specific application context and contained the tag `<repositories base-package="com.atreceno.it.diaulos" />` to enable the configuration of Spring data repositories via XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.springframework.org/schema/data/jpa"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

  <repositories
    base-package="com.atreceno.it.diaulos" />

</beans:beans>
```

Furthermore, two other files were created as a result of the command `repository jpa`, the interface `EventRepository.java` and the aspect `EventRepository_Roo_Jpa_Repository.aj`. The `EventRepository.java` file can be seen below:

```
package com.atreceno.it.diaulos.repository;

import com.atreceno.it.diaulos.domain.Event;
import org.sf.roo.addon.layers.repository.jpa.RooJpaRepository;

@RooJpaRepository(domainType = Event.class)
public interface EventRepository {
}
```

The code of `EventRepository_Roo_Jpa_Repository.aj` is shown below:

```
package com.atreceno.it.diaulos.repository;

import com.atreceno.it.diaulos.domain.Event;
import com.atreceno.it.diaulos.repository.EventRepository;
import org.sf.data.jpa.repository.JpaRepository;
import org.sf.data.jpa.repository.JpaSpecificationExecutor;
import org.sf.stereotype.Repository;

privileged aspect EventRepository_Roo_Jpa_Repository {

    declare parents: EventRepository extends
        JpaRepository<Event, Long>;

    declare parents: EventRepository extends
        JpaSpecificationExecutor<Event>;

    declare @type: EventRepository: @Repository;

}
```

This aspect contained the annotation `@Repository` indicating the interface `EventRepository` was indeed a *repository* (as defined in the Domain-Driven Design) and provided `DataAccessException` translation. In the same aspect, there were two interfaces being extended, `JpaRepository` and `JpaSpecificationExecutor`. Both interfaces are part of the Spring Data JPA (see Konda 2012; Gierke et al. 2012; Kainulainen 2012) and contain methods to find, save and delete objects. Additionally, `JpaSpecificationExecutor` accepts a search criteria with a `Specification` class, sort parameters and pagination.

3.2.3 Service Layer

3.2.3.1 Configuring a Service Layer

Roo also provided a way to define services to back the persistence layer. Even though it is possible to back entities based on the Active record pattern, only the entities with a persistence layer based on *repository* were exposed through a service layer:

```
roo> service --interface ~.service.EventService --entity
~.domain.Event
Created SRC_MAIN_JAVA/com/atreceno/it/diaulos/service
Created SRC_MAIN_JAVA/com/.../EventService.java
Created SRC_MAIN_JAVA/com/.../EventServiceImpl.java
Created SRC_MAIN_JAVA/com/.../EventService_Roo_Service.aj
Created SRC_MAIN_JAVA/com/.../EventServiceImpl_Roo_Service.aj
```

Roo created the inteface `EventService.java` and the corresponding implementation `EventServiceImpl.java`. The interface just contained the annotation `@RooService` with the name of the entity exposed by the service, `Event` in this case. The class `EventServiceImpl.java` is the implementation of the previous interface and this is where the business logic should go. Roo also created two aspects: `EventService_Roo_Service.aj` and `EventServiceImpl_Roo_Service.aj`. Here is the code for the first one:

```
package com.atreceno.it.diaulos.service;

import com.atreceno.it.diaulos.domain.Event;
import com.atreceno.it.diaulos.service.EventService;
import java.util.List;

privileged aspect EventService_Roo_Service {

    public abstract long EventService.countAllEvents();
    public abstract void EventService.deleteEvent(Event event);
    public abstract Event EventService.findEvent(Long id);
    public abstract List<Event> EventService.findAllEvents();
    public abstract List<Event> EventService.
        findEventEntries(int firstResult, int maxResults);
    public abstract void EventService.saveEvent(Event event);
    public abstract Event EventService.updateEvent(Event event);
}
```

It contained the method's signatures to create, find, save and delete events. This interface was backed by the aspect `EventServiceImpl_Roo_Service.aj`, which is shown in the next listing:

```
package com.atreceno.it.diaulos.service;

import com.atreceno.it.diaulos.domain.Event;
import com.atreceno.it.diaulos.repository.EventRepository;
import com.atreceno.it.diaulos.service.EventServiceImpl;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

privileged aspect EventServiceImpl_Roo_Service {

    declare @type: EventServiceImpl: @Service;

    declare @type: EventServiceImpl: @Transactional;

    @Autowired
    EventRepository EventServiceImpl.eventRepository;

    public long EventServiceImpl.countAllEvents() {
        return eventRepository.count();
    }

    public void EventServiceImpl.deleteEvent(Event event) {
        eventRepository.delete(event);
    }

    public Event EventServiceImpl.findEvent(Long id) {
        return eventRepository.findOne(id);
    }

    public List<Event> EventServiceImpl.findAllEvents() {
        return eventRepository.findAll();
    }

    public List<Event> EventServiceImpl.
        findEventEntries(int firstResult, int maxResults) {
            return eventRepository.findAll(
                new org.springframework.data.domain.PageRequest(
                    firstResult / maxResults, maxResults
                )
            ).getContent();
    }

    public void EventServiceImpl.saveEvent(Event event) {
        eventRepository.save(event);
```

```
    }

    public Event EventServiceImpl.updateEvent(Event event) {
        return eventRepository.save(event);
    }

}
```

This aspect contained the `@Service` annotation and it was detected automatically via *classpath* scanning because `@Service` is a specialisation of `@Component`. The aspect was also annotated with `@Transactional` and it is explained in the next section with more detail. Finally, the service was woven with the `EventRepository` to manage persistency.

3.2.3.2 Transaction Management

Transactions play an important role in an enterprise application, preventing data from being inconsistent. They have four well known properties, represented by the acronym ACID: *Atomicity*, *Consistency*, *Isolation* and *Durability*. Transaction management allows to group several operations into a single logical unit of work. If one of the operations fails, then the transaction is rolled back, providing atomicity and ensuring the data is consistent should the unexpected occur. They also support concurrency by preventing another concurrent transaction from interfering with a transaction that has not yet been completed (see Mak and Guruzu 2010, chap. 13).

In Spring, there are three different ways of configuring transactions. Two of them are declarative: using annotations or XML-based. The third method is programmatic, i.e. writing Java code. Programming transactions gives more fine-grained control over transactions but they are intrusive and JavaBeans are littered with transaction logic. For this reason the declarative approach is recommended (see Ho and Harrop 2012, 484) and is the one used in this study. Transactions are a cross-cutting concern so Spring uses Aspect-Oriented Programming (AOP) to declare them. Here is an example of how to configure a transaction in XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
    http://www.sf.org/schema/beans
```

```
    http://www.sf.org/schema/beans/spring-beans-3.2.xsd
    http://www.sf.org/schema/tx
    http://www.sf.org/schema/tx/spring-tx-3.2.xsd">
  <tx:advice id="txAdvice"
    transaction-manager="txManager">
    <tx:attributes>
      <tx:method name="save*"
        propagation="REQUIRED" />
      <tx:method name="find*"
        propagation="SUPPORTS"
        read-only="true" />
    </tx:attributes>
  </tx:advice>
  <aop:config>
    <aop:advisor
      pointcut="execution(* *..EventService.*(..))"
      advice-ref="txAdvice" />
  </aop:config>
  ...
</beans>
```

The element `<tx:advice>` defines an *advice* with transaction attributes. And the element `<aop:config>` defines which beans should be given advice, i.e. the *pointcuts*.

The `tx` namespace also provides the `<tx:annotation-driven>` element to declare transactions using annotations. The root application context file `applicationContext.xml` was shown in the previous section, but part of it is reproduced here for convenience:

```
<bean class="org.sf.orm.jpa.JpaTransactionManager"
  id="transactionManager">
  <property name="entityManagerFactory"
    ref="entityManagerFactory" />
</bean>
<tx:annotation-driven
  mode="aspectj" transaction-manager="transactionManager" />
<bean
  class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean"
  id="entityManagerFactory">
  <property name="persistenceUnitName" value="persistenceUnit" />
  <property name="dataSource" ref="dataSource" />
</bean>
```

Spring uses a transaction manager to deal with a platform-specific transaction implementation, in this case, a Java Persistence API (JPA) transaction

manager, which allows to use JPA local transactions, i.e. when there is only one persistence resource. There are other implementations to manage local transactions like `DataSourceTransactionManager` to use JDBC transactions, or `HibernateTransactionManager` to use Hibernate transactions. Depending on the application requirements, there might be more than one persistence resource, like a database, and a message oriented middleware. Spring support distributed transactions, a.k.a. global transactions, using a third party Java Transaction API (JTA) implementation, such as Atomikos.

Once the `<tx:annotation-driven>` has been included in the application context, the container will look for beans annotated with `@Transactional`, either at the class level or at the method level, and automatically advise them. When the entity is an active record, both the persistence methods and the @Transactional annotations are defined in the aspect {Entity}*Roo*Jpa_ActiveRecord.aj. If the entity has a service layer, the `@Transactional` annotation will be defined in the aspect `{Entity}_ServiceImpl_Roo_Service.aj`. The transaction attributes are defined by parameters of the annotation as described below:

- *Propagation behaviour*: It defines when a transaction is created or when an existing transaction can be used. The default value is `PROPAGATION_REQUIRED`, which indicates that the current method must run within a transaction, creating a new one if there are none in progress. The possible values of propagation are defined as constants in the `TransactionDefinition` interface.
- *Isolation level*: Concurrent transactions can lead to *dirty reads*[5] and *phantom reads*[6]. On the other hand, perfect isolation can impact performance. The isolation level defines to what degree a transaction may be impacted by other concurrent transactions. The possible values are also defined as constants in the `TransactionDefinition` interface.
- *Read-only*: This attribute allows the persistence mechanism to apply certain optimisations. By default, read-only is set to false.
- *Transaction timeout*: The transaction will be rolled back after a certain number of seconds.
- *Rollback rules*: The transaction will be rolled back on specific checked exceptions. By default, they are rolled back only on runtime exceptions.

Saving an `Event` entity made the `JpaTransactionManager` implementation produce the following logs:

```
Found thread-bound EntityManager [org.hibernate.ejb.EntityManager
Impl@108c70da] for JPA transaction
```

[5] A *dirty read* occurs when a transaction reads data that has been written by another transaction but not yet committed.

[6] A *phantom read* occurs when a transaction reads for the second time and finds data that was not there before because another transaction has added new data.

```
Creating new transaction with name [org.springframework.data.jpa.
repository.support.SimpleJpaRepository.save]:
PROPAGATION_REQUIRED,ISOLATION_DEFAULT; ''
Hibernate: update event set code=?, description=?, ...
Initiating transaction commit
Committing JPA transaction on EntityManager [org.hibernate.ejb.
EntityManagerImpl@108c70da]
```

The transaction is for the `save` method, which has inherited the default values from the class-level annotation. After the update operation, Spring's JPA transaction manager has issued a commit, which causes Hibernate to flush the persistence context (see Ho and Harrop 2012, chap. 13; Walls 2011, chap. 6)

3.2.4 Web Layer

3.2.4.1 The Basics

In the past few years, web technologies have evolved at a very fast pace. This is, in part, due to users demanding both responsive and interactive applications. Users want to access web applications from any device as well as interact with them. At the same time, architects want to build modular applications that are as loosely coupled as possible. Spring framework provides a web module to address these issues. It is based on the Model-View-Controller (MVC) pattern. The MVC pattern is one of the most popular practices when it comes to designing web applications. The principal idea behind the pattern is the separation of concerns principle and modularity. It defines three main components:

- *Model*: Represents the business data.
- *View*: Presents the data to the user in a specific format.
- *Controller*: Handles the request from the user, manipulates the model and sends the view back to the user.

There are some derivatives of this pattern, such as the Model-View-Presenter (MVP) and the Model-View-ViewModel (MVVM) patterns. Google Web Toolkit and Backbone.js are examples of the former while AngularJS and KnockoutJS are examples of the latter. Even though the MVC pattern was originally defined to be implemented in the server, modern web applications implement this pattern in the front-end (see Andrews 2013).

3.2.4.2 Setting up Spring MVC

As with the previous layers, Roo was used to set up Spring MVC (see Rimple, Penchikala, and Alex 2012, chap. 5):

```
roo> web mvc setup
```

The `web mvc setup` command added the necessary libraries to the project descriptor, including `spring-webmvc`, `jsp-api` and `tiles-jsp`. It also created a user interface based on JSPX (an XML-compliant version of JavaServer Pages defined in JSR-245) with support for i18n, layouts, and several views. The structure of the `webapp` folder is shown below:

```
$ tree src/main/webapp --charset=ASCII -L 3
src/main/webapp
|-- WEB-INF
|   |-- classes
|   |   |-- alt.properties
|   |   `-- standard.properties
|   |-- i18n
|   |   |-- application.properties
|   |   |-- messages.properties
|   |   `-- messages_es.properties
|   |-- layouts
|   |   |-- default.jspx
|   |   `-- layouts.xml
|   |-- spring
|   |   `-- webmvc-config.xml
|   |-- tags
|   |   |-- form
|   |   |-- menu
|   |   `-- util
|   |-- views
|   |   |-- countrys
|   |   |-- dataAccessFailure.jspx
|   |   |-- ...
|   |   `-- views.xml
|   `-- web.xml
|-- images
|   |-- add.png
|   |-- ...
|   `-- update.png
|-- selenium
|   |-- test-country.xhtml
|   |-- ...
|   `-- test-venue.xhtml
`-- styles
    |-- alt.css
    `-- standard.css
```

Following is an explanation of each subfolder:

- `WEB-INF/classes` contains properties to render the right style sheets (alternative or standard).
- `WEB-INF/i18n` contains localised property files:
  - `application.properties` contains business key-values, such as the name of the entities.
  - `messages.properties` contains generic key-values like *save*, *cancel*, *password*, and so on. The Roo command `web mvc language` can be used to support a different language.
- `WEB-INF/layouts` contains the layouts for use in Apache Tiles 2.2. Apache Tiles uses the Composite View Pattern to define a common layout of the page and reuse its components.
- `WEB-INF/spring/webmvc-config.xml` contains the Spring MVC configuration.
- `WEB-INF/tags` contains custom JSPX tags (`.tagx` extension) for use in forms, menus and pagination.
- `WEB-INF/views` contains the JSPX views of the application, such as `login.jspx` and `resourceNotFound.jspx`, as well as the business specific views.
- `WEB-INF/web.xml` is the Java EE Deployment descriptor defined in the JSR-315 specification, a.k.a. Java Servlet 3.0.
- `images` and `styles` contain the images and the style sheets of the application.

The `web.xml` deployment descriptor is shown in the next listing:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<web-app
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="2.5"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <display-name>diaulos</display-name>
  <description>Roo generated diaulos application</description>

  <context-param>
    <param-name>defaultHtmlEscape</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath*:META-INF/spring/applicationContext*.xml
```

```
  </param-value>
</context-param>

<filter>
  <filter-name>CharacterEncodingFilter</filter-name>
  <filter-class>
    org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter>
  <filter-name>HttpMethodFilter</filter-name>
  <filter-class>
    org.springframework.web.filter.HiddenHttpMethodFilter
  </filter-class>
</filter>
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>
    org.springframework.web.filter.DelegatingFilterProxy
  </filter-class>
</filter>
<filter>
  <filter-name>Spring
    OpenEntityManagerInViewFilter</filter-name>
  <filter-class>
    org.sf.orm.jpa.support.OpenEntityManagerInViewFilter
  </filter-class>
</filter>

<filter-mapping>
  <filter-name>CharacterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>HttpMethodFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
```

```
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>Spring
    OpenEntityManagerInViewFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- Creates Container shared by all Servlets and Filters -->
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>

<!-- Handles Spring requests -->
<servlet>
  <servlet-name>diaulos</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/spring/webmvc-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>diaulos</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<session-config>
  <session-timeout>10</session-timeout>
</session-config>

<error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/uncaughtException</location>
</error-page>

<error-page>
  <error-code>403</error-code>
  <location>/resourceNotFound</location>
```

```
  </error-page>

  <error-page>
    <error-code>404</error-code>
    <location>/resourceNotFound</location>
  </error-page>

</web-app>
```

There are two ways of creating an `ApplicationContext` for web applications:

- *Declaratively*: Using a context loader like `ContextLoaderListener`.
- *Programmatically*: Using one of the `ApplicationContext` implementations.

In this case, a `ContextLoaderListener` was used to create the application context based on the parameter `contextConfigLocation`, also defined in `web.xml` (see Rod Johnson 2013, 5). Along with the context loader, the following filters were defined and mapped to the URL `/*`:

- `CharacterEncodingFilter` specified a character encoding for those requests where the character encoding is not set.
- `HttpMethodFilter` allowed to use `PUT` and `DELETE` HTTP methods by using a hidden form field. This way, it was possible to build RESTful web APIs (see Goncalves 2013, chap. 15).
- `OpenEntityManagerInViewFilter` made JPA EntityManagers available via the current thread during the lifecycle of the request.

Additionally, the dispatcher servlet `diaulos` was configured according to the web application context `WEB-INF/spring/webmvc-config.xml` and mapped to the URL `/`.

The next step in the Spring MVC configuration was to create the controllers by issuing the Roo command `web mvc` (see Rimple, Penchikala, and Alex 2012, chap. 6):

```
roo> web mvc all --package ~.web
```

This triggered the creation of a controller per entity defined in the model and also the corresponding views and main menu. The generated class `EventController.java` is shown in the following listing:

```
package com.atreceno.it.diaulos.web;
```

```
import com.atreceno.it.diaulos.domain.Event;
import org.sf.roo.addon.web.mvc.controller.json.RooWebJson;
import org.sf.roo.addon.web.mvc.controller.scaffold.RooWebScaffold;
import org.sf.stereotype.Controller;
import org.sf.web.bind.annotation.RequestMapping;

@RequestMapping("/events")
@Controller
@RooWebScaffold(path = "events", formBackingObject = Event.class)
public class EventController {
}
```

The annotations of the controller are explained below:

- `@RequestMapping("/events")` allowed to map the URL `/events` to this controller.
- `@Controller` allowed this bean to be detected automatically through *class-path* scanning.
- `@RooWebScaffold` specified the subfolder in `WEB-INF/views` to store the corresponding views.

The class `EventController.java` was backed by the corresponding aspect, `EventController_Roo_Controller.aj`. A fragment of the code can be seen here:

```
package com.atreceno.it.diaulos.web;
// Import statements omitted

privileged aspect EventController_Roo_Controller {

    @Autowired
    EventService EventController.eventService;

    @Autowired
    ParticEventService EventController.particEventService;

    @Autowired
    PhaseService EventController.phaseService;

    @RequestMapping(method = RequestMethod.POST,
        produces = "text/html")
    public String EventController.create(@Valid Event event,
        BindingResult bindingResult, Model uiModel,
        HttpServletRequest httpServletRequest) {
```

```
        if (bindingResult.hasErrors()) {
            populateEditForm(uiModel, event);
            return "events/create";
        }
        uiModel.asMap().clear();
        eventService.saveEvent(event);
        return "redirect:/events/" + encodeUrlPathSegment(
            event.getId().toString(), httpServletRequest);
    }

    @RequestMapping(value = "/{id}", produces = "text/html")
    public String EventController.show(@PathVariable("id") Long
        id, Model uiModel) {

        uiModel.addAttribute("event", eventService.
            findEvent(id));
        uiModel.addAttribute("itemId", id);
        return "events/show";
    }

    @RequestMapping(produces = "text/html")
    public String EventController.list(@RequestParam(value =
        "page", required = false) Integer page, @RequestParam(
        value = "size", required = false) Integer size, Model
        uiModel) {

        if (page != null || size != null) {
            int sizeNo = size == null ? 10 : size.intValue();
            final int firstResult = page == null ? 0 :
                (page.intValue() - 1) * sizeNo;
            uiModel.addAttribute("events", eventService.
                findEventEntries(firstResult, sizeNo));
            float nrOfPages = (float) eventService.
                countAllEvents() / sizeNo;
            uiModel.addAttribute("maxPages", (int) ((nrOfPages >
                (int) nrOfPages || nrOfPages == 0.0) ? nrOfPages
                + 1 : nrOfPages));
        } else {
            uiModel.addAttribute("events", eventService.
                findAllEvents());
        }
        return "events/list";
    }
    // Other methods omitted
}
```

Each method had its own `@RequestMapping` annotation based on the HTTP method request and portions of the path. This way, each method defined a unique URL sub-pattern (see Deinum et al. 2012, chap. 5). For instance, a request of the path `/events` was mapped to the method `list()`. Notice how the `page` and `size` were extracted from the request with the annotation `@RequestParam`. The *model* in this case was the list of `events` and the *view* had access to them through the attribute `events`. The list of `events` was retrieved by using the service `eventService` previously injected by the Spring container. Finally, the view name `events/list` was returned. Because the `UrlBasedViewResolver` bean was configured in the web application context, the view `events/list` was actually `WEB-INF/views/events/list.jspx`, which is shown in the following listing:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<div xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:page="urn:jsptagdir:/WEB-INF/tags/form"
  xmlns:table="urn:jsptagdir:/WEB-INF/tags/form/fields"
  version="2.0">
  <jsp:directive.page contentType="text/html;charset=UTF-8" />
  <jsp:output omit-xml-declaration="yes" />
  <page:list id="pl_com_atreceno_it_diaulos_domain_Event"
    items="${events}" z="kLuEH7wBGqtJXanOYsJxMKiUmW8=">
    <table:table data="${events}"
      id="l_com_atreceno_it_diaulos_domain_Event"
      path="/events" z="KPofHFQjn/TYIWPEqBlMBFq/Wns=">
      <table:column
        id="c_com_atreceno_it_diaulos_domain_Event_code"
        property="code"
        z="B2ovJTb9Hrf7yPWuCXkXTR7MLXU=" />
      <table:column
        id="c_com_atreceno_it_diaulos_domain_Event_name"
        property="name"
        z="TQHJ7JMsjYlvdKe9m14o3I4OCxQ=" />
      <table:column
        id="c_com_atreceno_it_diaulos_domain_Event_description"
        property="description"
        z="5SOmGC4+Kvz8sZbgTiqh+Ud3UGk=" />
      <table:column
        id="c_com_atreceno_it_diaulos_domain_Event_gender"
        property="gender"
        z="n4C2UZxX2YlNnTSOmIZ/P/Dl1Ag=" />
      <table:column
        id="c_com_atreceno_it_diaulos_domain_Event_sport"
        property="sport"
        z="XrSpaMDxJFepvb574GSgoIdzaI8=" />
    </table:table>
```

```
  </page:list>
</div>
```

The namespace `page` and `table` are part of the custom library tags defined in `WEB-INF/tags`. The `z` attribute is a unique identifier that Roo uses internally to control a field. When a field is manually changed, the `z` attribute should be `z="user-managed"` to indicate that the field is now managed by the user.

The other important element in the Spring MVC's configuration is the web application context `webmvc-config.xml`:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.sf.org/schema/beans/spring-beans-3.1.xsd
    http://www.sf.org/schema/context
    http://www.sf.org/schema/context/spring-context-3.1.xsd
    http://www.sf.org/schema/mvc
    http://www.sf.org/schema/mvc/spring-mvc-3.1.xsd">

  <context:component-scan
    base-package="com.atreceno.it.diaulos"
    use-default-filters="false">
    <context:include-filter
      expression="org.springframework.stereotype.Controller"
      type="annotation" />
  </context:component-scan>
  <mvc:annotation-driven
    conversion-service="applicationConversionService" />
  <mvc:resources
    location="/, classpath:/META-INF/web-resources/"
    mapping="/resources/**" />
  <mvc:default-servlet-handler />
  <mvc:interceptors>
    <bean
      class="org.sf.web.servlet.theme.ThemeChangeInterceptor" />
    <bean
      class="org.sf.web.servlet.i18n.LocaleChangeInterceptor"
      p:paramName="lang" />
  </mvc:interceptors>
  <mvc:view-controller
```

```
  path="/login" />
<mvc:view-controller
  path="/" view-name="index" />
<mvc:view-controller
  path="/uncaughtException" />
<mvc:view-controller
  path="/resourceNotFound" />
<mvc:view-controller
  path="/dataAccessFailure" />
<bean
  class="org.sf.context.support.
  ReloadableResourceBundleMessageSource"
  id="messageSource"
  p:basenames="WEB-INF/i18n/messages,WEB-INF/i18n/application"
  p:fallbackToSystemLocale="false" />
<bean
  class="org.sf.web.servlet.i18n.CookieLocaleResolver"
  id="localeResolver" p:cookieName="locale" />
<bean
  class="org.sf.ui.context.support.ResourceBundleThemeSource"
  id="themeSource" />
<bean
  class="org.sf.web.servlet.theme.CookieThemeResolver"
  id="themeResolver" p:cookieName="theme"
  p:defaultThemeName="standard" />
<bean
  class="org.sf.web.servlet.handler.
  SimpleMappingExceptionResolver"
  p:defaultErrorView="uncaughtException">
  <property name="exceptionMappings">
    <props>
      <prop key=".DataAccessException">
        dataAccessFailure
      </prop>
      <prop key=".NoSuchRequestHandlingMethodException">
        resourceNotFound
      </prop>
      <prop key=".TypeMismatchException">
        resourceNotFound
      </prop>
      <prop key=".MissingServletRequestParameterException">
        resourceNotFound
      </prop>
    </props>
  </property>
</bean>
```

```
  <bean
    class="org.sf.web.multipart.commons.CommonsMultipartResolver"
    id="multipartResolver" />
  <bean
    class="org.sf.web.servlet.view.UrlBasedViewResolver"
    id="tilesViewResolver">
    <property name="viewClass"
      value="org.sf.web.servlet.view.tiles2.TilesView" />
  </bean>
  <bean
    class="org.sf.web.servlet.view.tiles2.TilesConfigurer"
    id="tilesConfigurer">
    <property name="definitions">
      <list>
        <value>/WEB-INF/layouts/layouts.xml
        </value>
        <value>/WEB-INF/views/**/views.xml
        </value>
      </list>
    </property>
  </bean>
  <bean
    class="com.atreceno.it.diaulos.web.
    ApplicationConversionServiceFactoryBean"
    id="applicationConversionService" />
</beans>
```

Following is an explanation of the main components (see Ho and Harrop 2012, chap. 14, 16–18; Walls 2011, chap. 7–9):

- `<context:component-scan>` allowed to register the `@Component` and specialisations, i.e. `@Repository`, `@Service` and `@Controller`, in the Spring container by scanning the *classpath* packages. In this case, because this was a web application context, only the `@Controller` components were registered. The rest of the stereotypes were registered in the root application context `applicationContext.xml`.
- `<mvc:annotation-driven>` element turned on support for mapping requests to controller methods and also registered default formatters and allowed bean validation (defined in specification JSR-303). A custom converter `applicationConversionService` was configured for use across all controllers.
- `<mvc:resources>` tag defined the location for static resources so they could be efficiently served. With this configuration, static resources could be served from both, the web application root and from the path `/META-INF/web-resources/` in any JAR on the *classpath*.

- `<mvc:default-servlet-handler>` tag allowed to forward requests that had not been mapped to the default servlet.
- `<mvc:interceptors>` allowed requests to be pre/post processed before/after handling. In this case, two interceptors were defined: `ThemeChangeInterceptor` and `LocaleChangeInterceptor`. They were applied to all `HandlerMapping` beans.
- `<mvc:view-controller>` defined a simple controller that immediately forwarded to a view because there was no logic to execute in the controller.

There were a few more bean definitions that allowed to load resource bundles; to store the theme and locale preferences in a cookie; and to define an exception and view resolvers.

Additionally, Roo installed the artifact `spring-js-resources` when setting up Spring MVC. This library contained several style sheets, functions to perform client side validation, as well as the library Dojo Toolkit, which offered Ajax support. They were located in the folder `META-INF/web-resources/` inside the jar, so they were available under the `resources` path. The scripts were included in the template `default.jspx` with the tag `<util:load-scripts />`. This way, the scripts were available from any page.

3.2.4.3 Adding Security

Spring Security is an authentication and authorisation framework for use in Spring applications (see Ben Alex 2013). Roo also offers support for Spring Security through the `security setup` command (see Rimple, Penchikala, and Alex 2012, chap. 8):

```
roo> security setup
Created SPRING_CONFIG_ROOT/applicationContext-security.xml
Created SRC_MAIN_WEBAPP/WEB-INF/views/login.jspx
Updated SRC_MAIN_WEBAPP/WEB-INF/views/views.xml
Updated ROOT/pom.xml [
added property 'security.version' = '3.1.0.RELEASE';
added dependencies
org.sf.security:spring-security-core:${security.version},
org.sf.security:spring-security-config:${security.version},
org.sf.security:spring-security-web:${security.version},
org.sf.security:spring-security-taglibs:${security.version}]
Updated SRC_MAIN_WEBAPP/WEB-INF/web.xml
Updated SRC_MAIN_WEBAPP/WEB-INF/spring/webmvc-config.xml
```

As a result, the following actions were taken:

- The necessary libraries were added to the Maven configuration file `pom.xml`.

- A new application context was generated to configure the security.
- A static view `login` was created and added to `views.xml` and `webmvc-config.xml`.
- A new filter `springSecurityFilterChain` was added to the deployment descriptor file.

The security application context is shown in the next listing:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans
  xmlns="http://www.springframework.org/schema/security"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.sf.org/schema/beans/spring-beans-3.1.xsd
    http://www.sf.org/schema/security
    http://www.sf.org/schema/security/spring-security-3.1.xsd">

  <!-- HTTP security configurations -->
  <http auto-config="true"
    use-expressions="true">
    <form-login
      login-processing-url="/resources/j_spring_security_check"
      login-page="/login"
      authentication-failure-url="/login?login_error=t" />
    <logout
      logout-url="/resources/j_spring_security_logout" />
    <!-- Configure URIs in the application -->
    <intercept-url
      pattern="/login/**" access="permitAll" />
    <intercept-url
      pattern="/resources/**" access="permitAll" />
    <intercept-url
      pattern="/**" method="GET"
      access="isAuthenticated()" />
    <intercept-url
      pattern="/**" access="hasRole('ROLE_ADMIN')" />
    <remember-me
      data-source-ref="dataSource"
      user-service-ref="jdbcUserService" />
  </http>
  <!-- Configure Authentication mechanism -->
  <authentication-manager
    alias="authenticationManager">
    <!-- SHA-256 values can be produced using
```

```
      'echo -n your_desired_password | sha256sum'
      (using normal *nix environments) -->
    <authentication-provider>
      <password-encoder
        hash="sha-256" />
      <jdbc-user-service
        id="jdbcUserService"
        data-source-ref="dataSource"
        users-by-username-query="select username, password,
          enabled from user where username = ?"
        authorities-by-username-query="select u.username, r.name
          from user u, role r, user_roles ur where u.id =
          ur.users and r.id = ur.roles and u.username = ?" />
    </authentication-provider>
    <authentication-provider>
      <password-encoder
        hash="sha-256" />
      <user-service>
        <user name="admin"
          password="d417a32def5666de310c785e770ded20cca348f..."
          authorities="ROLE_ADMIN" />
        <user name="user"
          password="d417a32def5666de310c785e770ded20cca348f..."
          authorities="ROLE_USER" />
      </user-service>
    </authentication-provider>
  </authentication-manager>
</beans:beans>
```

The security namespace was configured as the primary namespace so all tags without a prefix referred to the security schema. The main points are described below (see Walls 2011, chap. 9):

- The `<http>` element contained the HTTP configuration and included the following tags: `<form-login>` to configure a login form with username and password; `<logout>` to provide a logout filter; and `<intercept-url>` to configure the access attributes for a particular set of URLs.
- The `<authentication-manager>` element configured an authentication mechanism. In this case two providers were configured, both using the hashing algorithm `sha-256`: `<jdbc-user-service>` to store credentials in database tables; and `<user-service>` to create an in-memory `UserDetailsService` from a property file or, as in this case, from a list of users. This could be used as a backdoor in case the connection with the database failed.

To store the credentials in the database, it was necessary to create the corresponding tables and services (see Scarioni 2013):

```
roo> entity jpa --class ~.domain.security.User --equals
--activeRecord false --permitReservedWords
roo> entity jpa --class ~.domain.security.Role --equals
--activeRecord false --permitReservedWords
roo> focus --class ~.domain.security.User
roo> field string --fieldName username --notNull --sizeMin 3
--sizeMax 20
roo> field string --fieldName password --notNull --sizeMin 3
--sizeMax 65
roo> field boolean --fieldName enabled
roo> field set --fieldName roles --cardinality MANY_TO_MANY
--type ~.domain.security.Role
roo> focus --class ~.domain.security.Role
roo> field string --fieldName name --notNull --sizeMin 8
--sizeMax 20 --regexp ^ROLE_[A-Z]*
roo> field set --fieldName users --mappedBy roles
--cardinality MANY_TO_MANY --type ~.domain.security.User
roo> repository jpa --interface ~.repository.UserRepository
--entity ~.domain.security.User
roo> repository jpa --interface ~.repository.RoleRepository
--entity ~.domain.security.Role
roo> service --interface ~.service.UserService
--entity ~.domain.security.User
roo> service --interface ~.service.RoleService
--entity ~.domain.security.Role
```

Up to this point, the main components of the application have been described. The next section explains how to install the application in a cloud provider.

3.3 Cloud Deployment

In the past few years, *cloud computing* has changed the landscape of the IT industry and many organisations are turning to this relatively new concept because of two reasons: *Scalability* and *Cost*. Although cloud providers are constantly offering new products, there are three main service models:

- *Infrastructure as a Service (IaaS)*: The whole infrastructure, including storage, hardware, servers and networking, is outsourced to an external company. Typically, users are provided with virtual or physical servers that can be managed on demand. Amazon AWS, Joyent, Google Compute

Engine and Windows Azure are examples of *IaaS* providers. Most of *IaaS* providers offer solutions in the other two service models.

- *Platform as a Service (PaaS)*: Users are provided with all the necessary software to run their applications, including operative system, programming language, web server and databases. Cloud Foundry, Heroku, Nodejitsu and Google App Engine are examples of *PaaS* providers.
- *Software as a Service (SaaS)*: Cloud providers go one step further and offer not only the platform but the software itself, like Google Apps, Microsoft Office 365 and Rally Software.

Even though Amazon AWS is the biggest IaaS provider, the recent outages that occurred last summer in its data centre are making companies think of other alternatives. There are several factors to consider when choosing a cloud provider (see Millman 2012). Firstly, the service model, which depends on the nature of the business. If the software is not being offered by a *SaaS* provider, then the choices are *PaaS* or *IaaS*. In *PaaS*, there is less configuration to do, but it also implies less flexibility. Other factors to consider are the Service Level Agreement (SLA) and how the support is provided.

In this particular solution, two popular *PaaS* were used in combination with two database services:

- Pivotal CF with ClearDB.
- Heroku with Amazon RDS.

3.3.1 Pivotal CF with ClearDB

Pivotal CF is a commercially supported *PaaS* based on Cloud Foundry. It has been developed by VMware and Pivotal, the companies behind the Spring framework. At the time of writing, there were three options to deploy applications to Pivotal CF:

- Using SpringTool Suite (STS) with the Cloud Foundry plugin.
- Using Spring Roo with the Cloud Foundry add-on.
- Using a Ruby based command line tool called `cf`.

The plugin for STS was installed from the *Extensions* tab in the *Dashboard* and provided a graphical interface to manage the application in the cloud (see Figure 3.3).

Roo provided an add-on to manage the application in Cloud Foundry (see Rimple, Penchikala, and Alex 2012, chap. 13), but it was not used. The third option to interact with Pivotal CF was the command line tool `cf`. `cf` required at least *Ruby 1.9.3*, but *Mac OS X Lion* came with an older version so it needed to be upgraded first. Ruby's download website showed how to install *Ruby* with the package manager Homebrew:

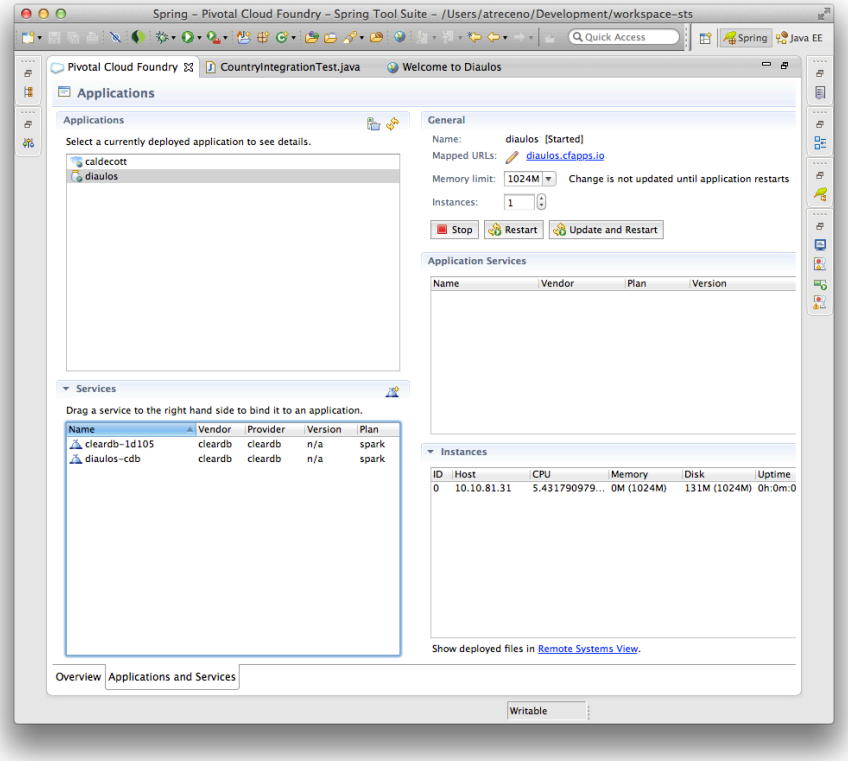

Figure 3.3: Cloud Foundry plugin for STS

```
$ ruby --version
ruby 1.8.7 (2012-02-08 patchlevel 358) [universal-darwin11.0]
$ brew install ruby
==> Summary
/usr/local/Cellar/ruby/2.0.0-p0: 877 files, 18M, built in 3.5 min
```

By default, `gem` installed binaries were placed into `/usr/local/opt/ruby/bin`, so the folder was added to the `PATH` variable:

```
$ head -n1 /etc/paths
/usr/local/opt/ruby/bin
$ ruby --version
ruby 2.0.0p0 (2013-02-24 revision 39474) [x86_64-darwin11.4.2]
```

Once, *Ruby* was upgraded, `cf` could be installed following the instructions from the Cloud Foundry's documentation website:

```
$ gem install cf
$ cf target api.run.pivotal.io
Setting target to https://api.run.pivotal.io... OK
$ cf login
target: https://api.run.pivotal.io
Email> agustin@gast.it.uc3m.es
Password> **********
Authenticating... OK
1: development
2: production
3: staging
Space> 1
Switching to space development... OK
```

The command `cf push` was used to upload the application to the server:

```
$ mvn package
$ cf push --path target/diaulos-0.1.0.BUILD-SNAPSHOT.war
(some lines omitted)
Saving to manifest.yml... OK
Uploading diaulos... OK
Preparing to start diaulos... OK
-----> Downloaded app package (49M)
-----> Downloading OpenJDK 1.7.0_21 JRE
       Expanding JRE to .java (1.0s)
-----> Downloading Auto Reconfiguration 0.7.1
       Modifying /WEB-INF/web.xml for Auto Reconfiguration
```

```
-----> Downloading Tomcat 7.0.42
       Expanding Tomcat to .tomcat (0.1s)
       Downloading Buildpack Tomcat Support 1.1.1
Checking status of app 'diaulos'...
  0 of 1 instances running (1 starting)
  1 of 1 instances running (1 running)
Push successful! App 'diaulos' available at http://diaulos.cfapps.io
```

When deploying the application for the first time, the command `cf` asked a few questions about the number of instances, amount of memory, domain name, and so on. These options were saved to the file `manifest.yml`:

```
$ cat manifest.yml
---
applications:
- name: diaulos
  memory: 1G
  instances: 1
  host: diaulos
  domain: cfapps.io
  path: target/diaulos-0.1.0.BUILD-SNAPSHOT.war
  services:
    diaulos-cdb:
      label: cleardb
      provider: cleardb
      version: n/a
      plan: spark
```

The previous listing shows that a ClearDB service was bound to the application. Once the application was uploaded, it could be managed from the console with `cf start`, `cf stop` and `cf scale` and also monitored with `cf logs`, `cf events` and `cf crashlogs`. The complete set of commands was available by issuing the command `cf` without any parameters. Additionally, Pivotal CF provided a web interface where it was possible to manage several environments as shown in Figure 3.4.

Before using the application, the database had to be populated. The URL to access the database was available in the file `logs/env.log`:

```
$ cf file diaulos logs/env.log
Getting file contents... OK
(some lines omitted)
TMPDIR=/home/vcap/tmp
VCAP_APP_PORT=61963
JAVA_OPTS=-Xmx1024m -Xms1024m -Dhttp.port=61963
```

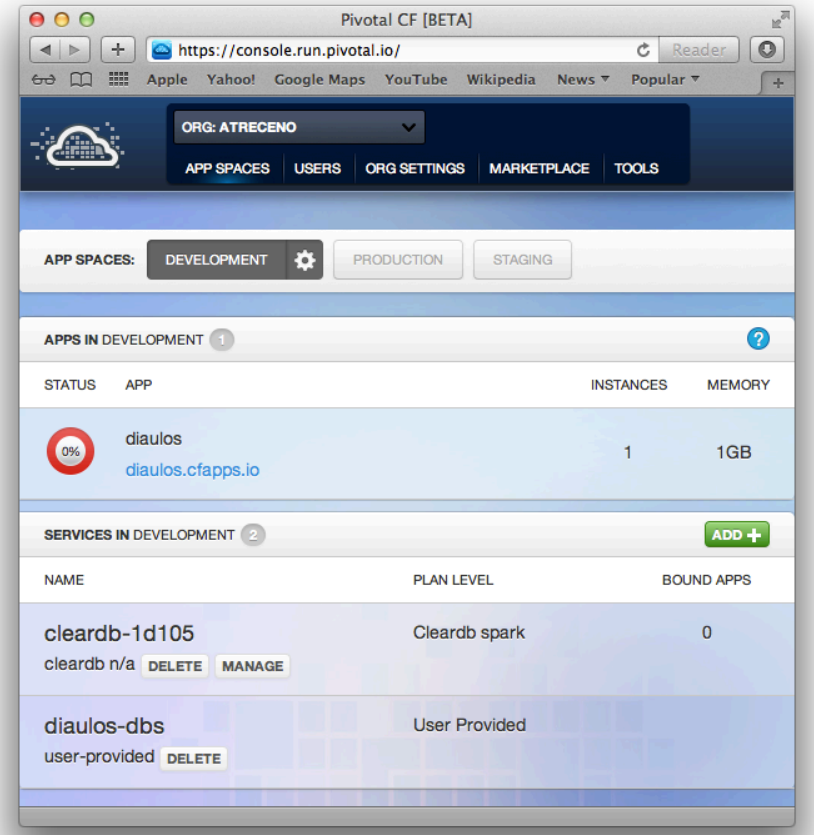

Figure 3.4: Pivotal CF web interface

```
USER=vcap
HOME=/home/vcap/app
PORT=61963
VCAP_APP_HOST=0.0.0.0
DATABASE_URL=mysql2://bbc5085b02f504:d7394f60@us-cdbr-east-04.
cleardb.com:3306/ad_42ff4a1a30ee7df?reconnect=true
MEMORY_LIMIT=1024m
```

Using the previous information, it was possible to use `mysqldump` to export the local database and then `mysql` to import the data into the production database:

```
$ mysqldump -u ulises -p diaulos_db > diaulos_dump.sql
$ mysql -h us-cdbr-east-04.cleardb.com -u bbc5085b02f504 -p \
ad_42ff4a1a30ee7df < diaulos_dump.sql
```

Once the database was populated, the application was ready to use (see Figure 3.5).

Unfortunately, ClearDB's *Spark* plan was the only MySQL service provider available in Pivotal CF, which had some limitations. For instance, it only offered 5 MB of capacity and it was not possible to rename the database or change the username or password. The complete list of athletes and sporting events (not including the results) was 3.7 MB but once that data was imported into MySQL database, it became 35 MB, exceeding the 5 MB capacity limit of ClearDB's *Spark* plan:

```
$ sudo du -h /usr/local/mysql/data/diaulos_db
4.0K    /usr/local/mysql/data/diaulos_db
$ du -h diaulos_dump.sql
3.7M    diaulos_dump.sql
$ mysql -u root -p diaulos_db < diaulos_dump.sql
$ sudo du -h /usr/local/mysql/data/diaulos_db
 35M    /usr/local/mysql/data/diaulos_db
```

For this reason, its use was discouraged and it was not taken into consideration for the load tests.

3.3.2 Heroku with Amazon RDS

Heroku was one of the first cloud platforms available. Heroku, like Pivotal CF, provides a command line tool and a web interface to manage the application (see Figure 3.6).

After installing Heroku Toolbelt, which included the Heroku client, it was possible to authenticate by issuing the command `heroku login`. To be able to

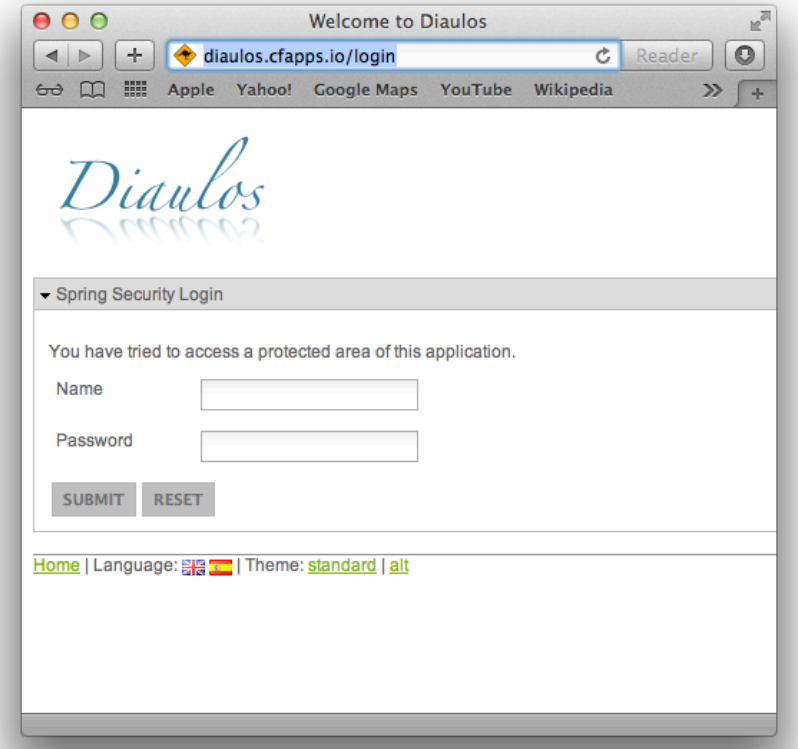

Figure 3.5: Diaulos deployed to Pivotal CF

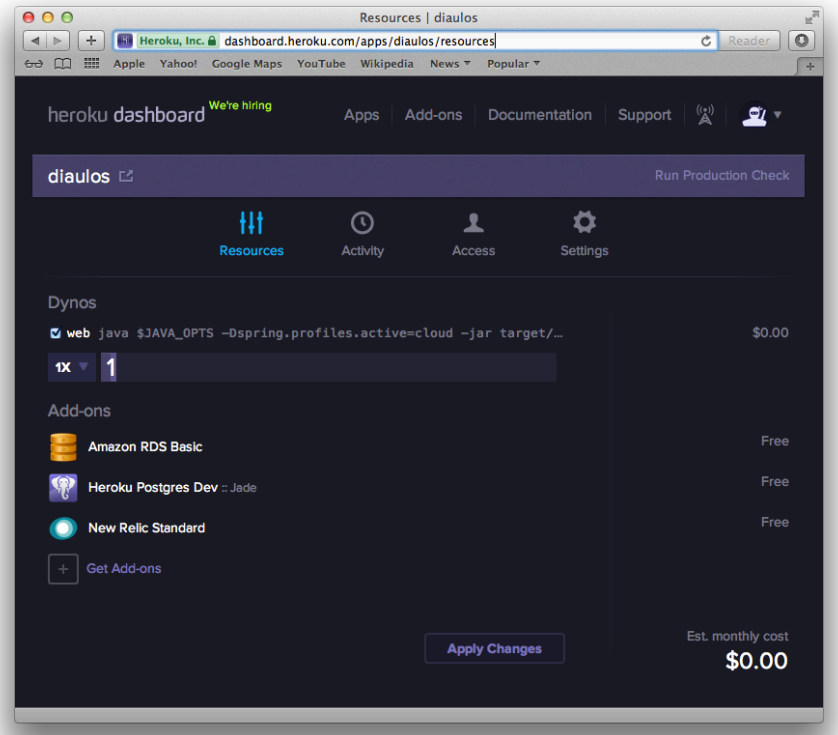

Figure 3.6: Heroku Dashboard - Diaulos

run the application in Heroku, it was necessary to install Jetty Runner, which allowed to launch the application in a Jetty container by using `java` from the command line. It was also possible to use Webapp Runner, which used Tomcat instead of Jetty. The `maven-dependency-plugin` was used to copy the JAR file to the `target` directory during the `package` phase, as follows:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.8</version>
  <executions>
    <execution>
      <id>copy-jetty-runner</id>
      <phase>package</phase>
      <goals>
        <goal>copy</goal>
      </goals>
      <configuration>
        <artifactItems>
          <artifactItem>
            <groupId>org.mortbay.jetty</groupId>
            <artifactId>jetty-runner</artifactId>
            <version>8.1.13.v20130916</version>
            <destFileName>jetty-runner.jar</destFileName>
          </artifactItem>
        </artifactItems>
      </configuration>
    </execution>
  </executions>
</plugin>
```

The command that Heroku used to launch the application was defined in the file `Procfile`:

```
$ cat Procfile
web: java $JAVA_OPTS -Dspring.profiles.active=cloud -jar \
target/dependency/jetty-runner.jar --port $PORT target/*.war
```

Along with the file `system.properties` which specified the Java version:

```
$ cat system.properties
java.runtime.version=1.7
```

The command `heroku create` was issued before deploying the application to Heroku:

```
$ heroku create --region eu
Creating sea-9796... done, region is eu
http://sea-9796.herokuapp.com/ | git@heroku.com:sea-9796.git
Git remote heroku added
$ git remote -v
heroku  git@heroku.com:sea-9796.git (fetch)
heroku  git@heroku.com:sea-9796.git (push)
origin  git@github.com:atreceno/diaulos.git (fetch)
origin  git@github.com:atreceno/diaulos.git (push)
```

It was possible to change the application name and Git URL from the dashboard, but the latter had to be in concordance with the remote Git repository:

```
$ git remote rm heroku
$ git remote add heroku git@heroku.com:diaulos.git
$ git remote -v
heroku  git@heroku.com:diaulos.git (fetch)
heroku  git@heroku.com:diaulos.git (push)
origin  git@github.com:atreceno/diaulos.git (fetch)
origin  git@github.com:atreceno/diaulos.git (push)
```

The next step was to create the database. This time, using Amazon RDS which allowed to use and scale a MySQL database server on top of Amazon EC2. Services could be managed from the *AWS Management Console* (see Figure 3.7), and after the database was created, the `DATABASE_URL` was configured in Heroku as follows:

```
$ heroku addons:add amazon_rds url=\
mysql2://hackney:london@rdshostname.amazonaws.com/diaulos_db
Adding amazon_rds to appname...Done.
$ heroku config:get DATABASE_URL
mysql2://hackney:london@rdshostname.amazonaws.com/diaulos_db
```

The environment variable `DATABASE_URL` was accessed from the application context as follows:

```
<!-- Production Profile -->
<beans profile="cloud">
  <bean class="java.net.URI" id="dbUrl">
    <constructor-arg
      value="#{systemEnvironment['DATABASE_URL']}" />
  </bean>
  <bean id="dataSource" class="com.jolbox.bonecp.BoneCPDataSource"
    destroy-method="close">
```

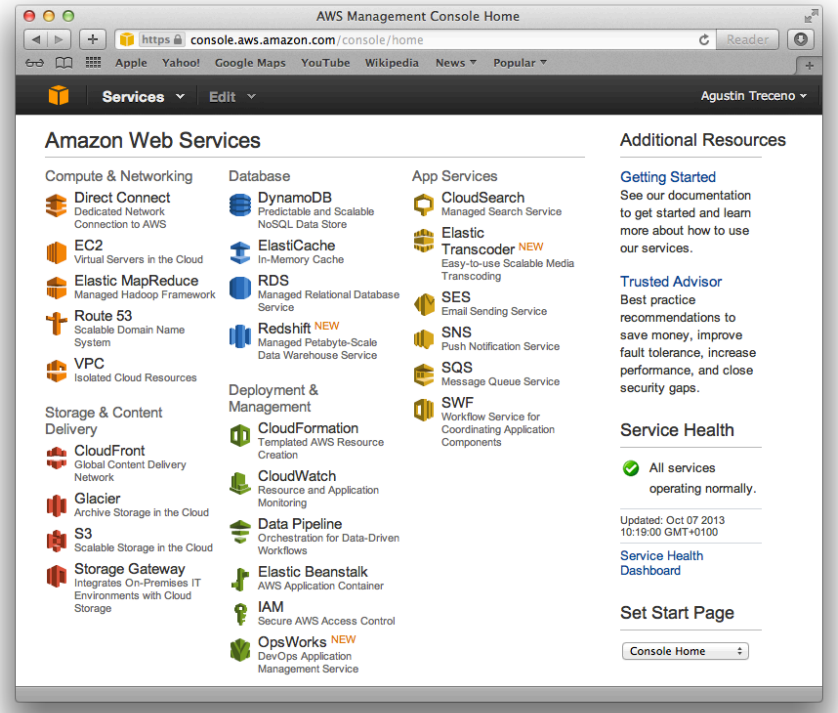

Figure 3.7: AWS Management Console

```
    <property name="driverClass" value="com.mysql.jdbc.Driver" />
    <property name="jdbcUrl" value="#{ 'jdbc:mysql://' +
      @dbUrl.getHost() + @dbUrl.getPath() }" />
    <property name="username"
      value="#{ @dbUrl.getUserInfo().split(':')[0] }" />
    <property name="password"
      value="#{ @dbUrl.getUserInfo().split(':')[1] }" />
    <property name="idleConnectionTestPeriodInMinutes"
      value="60" />
    <property name="idleMaxAgeInMinutes" value="240" />
    <property name="maxConnectionsPerPartition" value="30" />
    <property name="minConnectionsPerPartition" value="10" />
    <property name="partitionCount" value="3" />
    <property name="acquireIncrement" value="5" />
    <property name="statementsCacheSize" value="100" />
  </bean>
  <bean class="org.springframework.orm.jpa.LocalContainerEntity
    ManagerFactoryBean" id="entityManagerFactory">
    <property name="persistenceUnitName" value="persistenceUnit"/>
    <property name="dataSource" ref="dataSource" />
    <property name="jpaProperties">
      <props>
        <prop key="hibernate.dialect">
          org.hibernate.dialect.MySQL5InnoDBDialect
        </prop>
      </props>
    </property>
  </bean>
</beans>
```

The production profile uses `BoneCPDataSource` from BoneCP which had a better performance for concurrent users than `BasicDataSource` from Apache Commons DBCP and `ComboPoolsedDataSource` from the C3P0 project.

Additionally, Amazon RDS worked behind a firewall and it was necessary to add a rule to be able to connect to the database from a MySQL client and import the data:

```
$ mysqldump -u bricklane -p diaulos_db > diaulos_dump.sql
$ mysql -h diaulos.eu-west-1.rds.amazonaws.com -u hackney -p \
diaulos_db < diaulos_dump.sql
```

Once the application was created and the database populated, it could be deployed by issuing the command `git push` as follows:

```
$ git push heroku master
```

```
Counting objects: 27, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (16/16), done.
Writing objects: 100% (16/16), 1.46 KiB, done.
Total 16 (delta 8), reused 0 (delta 0)

-----> Java app detected
-----> Installing OpenJDK 1.7...done
-----> Installing settings.xml... done
-----> executing /app/tmp/repo.git/.cache/.maven/bin/mvn -B
-Duser.home=/tmp/build_0fe01f5a-3b38-408b-8731-616b27ffe421
-Dmaven.repo.local=/app/tmp/repo.git/.cache/.m2/repository
-s /app/tmp/repo.git/.cache/.m2/settings.xml
-DskipTests=true clean install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building diaulos 0.1.0.BUILD-SNAPSHOT
[INFO] ------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.4.1:clean (default-clean)
[INFO]
[INFO] --- aspectj-maven-plugin:1.2:compile (default)
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (resources)
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 18 resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile)
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- aspectj-maven-plugin:1.2:test-compile (default)
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (testResources)
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /tmp/.../resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:testCompile (testCompile)
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12:test (default-test)
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ diaulos ---
[INFO] Packaging webapp
[INFO] Assembling webapp [diaulos] in [.../diaulos-0.1.0.BUILD]
```

```
[INFO] Processing war project
[INFO] Copying webapp resources [/tmp/.../src/main/webapp]
[INFO] Webapp assembled in [833 msecs]
[INFO] Building war: /tmp/.../diaulos-0.1.0.BUILD-SNAPSHOT.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO]
[INFO] --- maven-dependency-plugin:2.8:copy (copy-jetty-runner)
[INFO] Configured Artifact: org.mortbay.jetty:jetty-runner:8.1.13
.v20130916:jar
[INFO] Copying jetty-runner-8.1.13.v20130916.jar to /tmp/build_
0fe01f5a-3b38-408b-8731-616b27ffe421/target/dependency/jetty-
runner.jar
[INFO]
[INFO] --- maven-dependency-plugin:2.8:copy-dependencies (copy-
newrelic-agent) @ diaulos ---
[INFO] Copying newrelic-agent-2.21.4.jar to /tmp/build_0fe01f5a-
3b38-408b-8731-616b27ffe421/target/dependency/newrelic-agent.jar
[INFO]
[INFO] --- maven-failsafe-plugin:2.16:integration-test (selenium-
tests) @ diaulos ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-failsafe-plugin:2.16:verify (verify) @ diaulos
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-install-plugin:2.3.1:install (default-install)
[INFO] Installing /tmp/build_0fe01f5a-3b38-408b-8731-616b27ffe421
/target/diaulos-0.1.0.BUILD-SNAPSHOT.war to /app/tmp/repo.git/.ca
che/.m2/repository/com/atreceno/it/diaulos/diaulos/0.1.0.BUILD-SN
APSHOT/diaulos-0.1.0.BUILD-SNAPSHOT.war
[INFO] Installing /tmp/build_0fe01f5a-3b38-408b-8731-616b27ffe421
/pom.xml to /app/tmp/repo.git/.cache/.m2/repository/com/atreceno/
it/diaulos/diaulos/0.1.0.BUILD-SNAPSHOT/diaulos-0.1.0.BUILD-SNAPS
HOT.pom
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 24.113s
[INFO] Finished at: Sun Oct 06 19:53:16 UTC 2013
[INFO] Final Memory: 17M/468M
[INFO] ------------------------------------------------------------
-----> Discovering process types
       Procfile declares types -> web

-----> Compiled slug size: 147.2MB
-----> Launching... done, v10
```

```
-----> Deploy hooks scheduled, check output in your logs
       http://diaulos.herokuapp.com deployed to Heroku

To git@heroku.com:diaulos.git
   3ba0f1e..6f6e692  master -> master
```

To instruct Heroku to execute the process type specified in `Procfile` the command `heroku ps:scale` was used:

```
$ heroku ps:scale web=1
Scaling web dynos... done, now running 1
$ heroku ps
=== web (1X): `java $JAVA_OPTS -Dspring.profiles.active=cloud \
-jar target/dependency/jetty-runner.jar --port $PORT target/*.war`
web.1: up 2013/10/07 08:34:13 (~ 52m ago)
```

Heroku then used one *dyno* to run the `web` process type. A dyno was a lightweight container running a single user-specific command and it came in two different sizes: 1X and 2X, with 512 MB and 1024 MB, respectively. If the process exceeded its quota, the memory was swapped out to disk, which degraded the performance of the application. If the memory of the process kept growing and reached three times its quota, the process was restarted.

After setting one `web dyno`, the application was ready to receive requests:

```
$ heroku logs --tail
2013-10-07T07:34:12.103939+00:00 app[web.1]: 2013-10-07 07:34:12,
103 [main] INFO  org.apache.tiles.context.AbstractTilesApplicatio
nContextFactory - Initializing Tiles2 application context. . .
2013-10-07T07:34:12.191720+00:00 app[web.1]: 2013-10-07 07:34:12,
191 [main] INFO  org.apache.tiles.context.AbstractTilesApplicatio
nContextFactory - Finished initializing Tiles2 application context.
2013-10-07T07:34:12.719643+00:00 app[web.1]: 2013-10-07 07:34:12,
719 [main] INFO  org.apache.tiles.access.TilesAccess - Publishing
TilesContext for context: org.springframework.web.servlet.view.ti
les2.SpringTilesApplicationContextFactory$SpringWildcardServletTi
lesApplicationContext
2013-10-07T07:34:12.746808+00:00 app[web.1]: 2013-10-07 07:34:12,
746 [main] INFO  org.springframework.web.servlet.DispatcherServle
t - FrameworkServlet 'diaulos': initialization completed in 5666ms
2013-10-07T07:34:12.954612+00:00 app[web.1]: 2013-10-07 07:34:12.
954:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.
0.0.0:7022
2013-10-07T07:34:12.853857+00:00 heroku[web.1]: State changed
from starting to up
```

The following logs, show a user logging into the application and viewing the list of sports:

```
$ heroku logs --tail
2013-10-07T08:45:17.861173+00:00 heroku[router]: at=info method=
GET path=/login host=diaulos.herokuapp.com fwd="86.160.254.197"
dyno=web.1 connect=0ms service=37ms status=200 bytes=4012
2013-10-07T08:46:03.174548+00:00 heroku[router]: at=info method=
POST path=/resources/j_spring_security_check host=diaulos.
herokuapp.com fwd="86.160.254.197" dyno=web.1 connect=0ms
service=188ms status=302 bytes=0
2013-10-07T08:46:04.385863+00:00 heroku[router]: at=info method=
GET path=/ host=diaulos.herokuapp.com fwd="86.160.254.197"
dyno=web.1 connect=0ms service=1178ms status=200 bytes=5748
2013-10-07T08:46:20.682029+00:00 heroku[router]: at=info method=
GET path=/sports?page=1&size=10 host=diaulos.herokuapp.com
fwd="86.160.254.197" dyno=web.1 connect=0ms service=4347ms
status=200 bytes=15210
```

3.4 Testing

A web application can be tested at several levels (see Rimple, Penchikala, and Alex 2012, chap. 9): using isolated, out-of-container unit tests; in-container integration tests; and black-box functional tests. Ho and Harrop (2012) describe enterprise testing in more detail and differentiate 8 different categories:

- Logic Unit Test: Allows to test the application at the method level with all other dependencies being *mocked*.
- Integration Unit Test: Checks the interaction of several classes at different layers of the application.
- Front-end Unit Test: Tests the user interface of the application helped by an external tool to automate the process.
- Continuous Build and Code Quality Test: The application should be built on a regular bases using Continuous Integration (CI) tools, e.g. Jenkins or Travis; and code quality tools, e.g. Gerrit.
- System Integration Test: Verifies the integration among the different systems used by the application in an environment similar to production.
- Functional Test: A black-box testing using the Graphical User Interface (GUI) and verifying the results are in concordance with the use cases and business rules.
- System Quality Test: The purpose of the system quality test, a.k.a. non-functional test, is to ensure the application works under a typical work load and beyond.

- User Acceptance Test: Requires the presence of the end user to test the application in normal working condition.

Depending on the nature of the application, some of these categories can be merged into one or removed completely. For instance, the front-end unit testing could validate the functional requirements as well as the correct integration of the systems in the given environment. For the *Diaulos* application the testing strategy was as follows:

- Logic Unit Tests were done at the service layer, mocking all the other dependencies.
- Integration Unit Tests were done at the service layer as well, but this time using the actual repository layer instead of mocking it.
- Front-end Unit Tests were done with the tool Selenium. As previously mentioned, these tests cover the system integration tests and the functional tests.
- Non-functional tests were done with the command line tool `siege`.

Several tools were used to monitor the different parts of the application, including Java Visual VM, Spring Insight, MySQL Workbench, New Relic and Activity Monitor.

3.4.1 Logic Unit Tests

One of the most popular approaches, especially in unit tests, is Test-Driven Development (TDD). TDD is part of the Agile software development methodology and it promotes the implementation of the test cases before the actual implementation of the code to pass the tests. Stephens and Rosenberg (2010) describes a different approach based on writing tests driven by the design instead of writing code driven by the tests. TDD was used to test the service layer of the *Diaulos* application. As an example, one of the methods of the service `EventService` was tested. This service had a dependency on the `EventRepository` but because this was a logic unit test, this dependency was *mocked* by a popular framework known as Mockito. The following listing shows the file `EventServiceImplTest.java`:

```
package com.atreceno.it.diaulos.service;

import java.util.ArrayList;
import java.util.List;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
```

```
import org.junit.runners.JUnit4;
import org.mockito.Mockito;
import org.springframework.test.util.ReflectionTestUtils;

import com.atreceno.it.diaulos.domain.Event;
import com.atreceno.it.diaulos.repository.EventRepository;

@RunWith(JUnit4.class)
public class EventServiceImplTest {

    private List<Event> events = new ArrayList<Event>();

    /**
     * EventRepository to mock
     */
    private EventRepository eventRepository;

    /**
     * Initialize list of Events
     */
    @Before
    public void initEvents() {
        for (int i = 0; i < 5; i++) {
            Event event = new Event();
            event.setCode("code_" + i);
            event.setName("name_" + i);
            event.setDescription("description_" + i);
            events.add(event);
        }
    }

    /**
     * Test findAllEvents method
     */
    @Test
    public void testFindAllEvents() {

        // Mock EventRepository
        eventRepository = Mockito.mock(EventRepository.class);
        Mockito.when(eventRepository.findAll())
            .thenReturn(events);

        // Create EventService and inject the mocked repository
        EventService eventService = new EventServiceImpl();
        ReflectionTestUtils.setField(eventService,
            "eventRepository", eventRepository);
```

```
        // Use the Service
        List<Event> result = eventService.findAllEvents();

        // Assert
        Assert.assertNotNull(result);
        Assert.assertEquals(5, result.size());
        for (int i = 0; i < 5; i++) {
            Event event = result.get(i);
            Assert.assertEquals("code_" + i, event.getCode());
            Assert.assertEquals("name_" + i, event.getName());
            Assert.assertEquals("description_" + i,
                event.getDescription());
        }

    }

}
```

The interesting point in the previous listing is the use of the methods `mock()`, `when()` and `thenReturn()` of the Mockito framework. This way, when the service used the method `findAll()` of the repository, the mocked object returned the list of events that was previously initialised in the method annotated with `@Before`. This kind of testing becomes really handy when the service layer uses a third-party system like a weather provider because the external system can be easily mocked (see Tahchiev and Massol 2011, chap. 7).

3.4.2 Integration Unit Tests

Roo provides an easy way of testing entities with the command `test integration` or passing the parameter `--testAutomatically` when creating an entity. The class `RaceIntegrationTest.java` is shown in the next listing as an example:

```
package com.atreceno.it.diaulos.domain;

import org.junit.Test;
import org.springframework.roo.addon.test.RooIntegrationTest;

@RooIntegrationTest(entity = Race.class)
public class RaceIntegrationTest {

    @Test
    public void testMarkerMethod() {
```

```
    }
}
```

The class was backed by two aspects:

- `RaceIntegrationTest_Roo_Configurable.aj`, which contained the annotation `@Configurable` so it could be managed by the container.
- `RaceIntegrationTest_Roo_IntegrationTest.aj`, which contained the actual methods under test.

Part of the code of the second aspect is shown below:

```
package com.atreceno.it.diaulos.domain;

// import statements omitted.

privileged aspect RaceIntegrationTest_Roo_IntegrationTest {

    declare @type: RaceIntegrationTest: @RunWith(
        SpringJUnit4ClassRunner.class);

    declare @type: RaceIntegrationTest: @ContextConfiguration(
        locations = "classpath*:/META-INF/spring/
            applicationContext*.xml");

    declare @type: RaceIntegrationTest: @Transactional;

    @Autowired
    RaceDataOnDemand RaceIntegrationTest.dod;

    @Autowired
    RaceService RaceIntegrationTest.raceService;

    @Autowired
    RaceRepository RaceIntegrationTest.raceRepository;

    @Test
    public void RaceIntegrationTest.testCountAllRaces() {
        Assert.assertNotNull("Data on demand for 'Race' failed to
            initialize correctly", dod.getRandomRace());
        long count = raceService.countAllRaces();
        Assert.assertTrue("Counter for 'Race' incorrectly
            reported there were no entries", count > 0);
    }
```

```
    @Test
    public void RaceIntegrationTest.testFindRace() {
        Race obj = dod.getRandomRace();
        Assert.assertNotNull("Data on demand for 'Race' failed to
            initialize correctly", obj);
        Long id = obj.getId();
        Assert.assertNotNull("Data on demand for 'Race' failed to
            provide an identifier", id);
        obj = raceService.findRace(id);
        Assert.assertNotNull("Find method for 'Race' illegally
            returned null for id '" + id + "'", obj);
        Assert.assertEquals("Find method for 'Race' returned the
            incorrect identifier", id, obj.getId());
    }

    @Test
    public void RaceIntegrationTest.testFindRaceEntries() {
        Assert.assertNotNull("Data on demand for 'Race' failed to
            initialize correctly", dod.getRandomRace());
        long count = raceService.countAllRaces();
        if (count > 20) count = 20;
        int firstResult = 0;
        int maxResults = (int) count;
        List<Race> result = raceService.findRaceEntries(
            firstResult, maxResults);
        Assert.assertNotNull("Find entries method for 'Race'
            illegally returned null", result);
        Assert.assertEquals("Find entries method for 'Race'
            returned an incorrect number of entries", count,
            result.size());
    }

// Some methods were omitted.

}
```

As shown in the previous listing, three dependencies were injected:

- `RaceDataOnDemand` to create instances of the entity `Race`
- `RaceService` to make use of the service under test.
- `RaceRepository` to flush all pending changes to the database.

To run all the tests together, the maven command `mvn test` was issued as follows:

```
$ mvn -e test
[INFO] Error stacktraces are turned on.
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building diaulos 0.1.0.BUILD-SNAPSHOT
[INFO] ------------------------------------------------------------
[INFO]
[INFO] --- aspectj-maven-plugin:1.2:compile (default) @ diaulos -
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources @ diaulos ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 19 resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile @ diaulos ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- aspectj-maven-plugin:1.2:test-compile @ diaulos ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources @ diaulos ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory .../src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:testCompile @ diaulos ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12:test @ diaulos ---
[INFO] Surefire report directory: .../target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------

Results :

Tests run: 128, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 39.710s
[INFO] Finished at: Tue Sep 10 22:04:18 BST 2013
[INFO] Final Memory: 7M/81M
[INFO] ------------------------------------------------------------
```

The maven-surefire-plugin was used to execute the unit tests and generate the reports. By default, the plugin tests all classes with the suffixes `Test` or `TestCase` and the prefix `Test`. It took around 40 seconds to complete the tests and all 128 tests were successful.

3.4.3 Front-end Unit Tests

For the front-end unit testing, a tool called Selenium was used to automate the tests. Selenium interacts with the Graphical User Interface (GUI) and inspects the result treating the application as a black-box. Roo provides the command `selenium test` with the name of the controller as a parameter. When this command was run, it added the plugin `selenium-maven-plugin` and its dependency `selenium-server` to the `pom.xml` as shown below:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>selenium-maven-plugin</artifactId>
  <version>2.3</version>
  <configuration>
    <browser>*firefox</browser>
    <suite>src/main/webapp/selenium/test-suite.xhtml</suite>
    <results>${project.build.directory}/selenium.html</results>
    <startURL>http://localhost:4444/</startURL>
  </configuration>
  <executions>
    <execution>
      <id>run-selenium-tests</id>
      <phase>integration-test</phase>
      <goals>
        <goal>selenese</goal>
      </goals>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-server</artifactId>
      <version>2.25.0</version>
    </dependency>
  </dependencies>
</plugin>
```

The test suite and test cases in XHTML format were located in the folder `webapp/WEB-INF/selenium`. The tests were run with the Maven command `mvn`

`selenium:selenese`. It was also possible to run them with the command `mvn integration-test` because they were included in the `integration-test` phase, as shown in the previous listing. Unfortunately, Roo did not implement the login test case and it did not take into consideration the JSR-303 validation of the entities. The test case `test-sport.xhtml` shown below demonstrates this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<html xmlns="http://www.w3.org/1999/xhtml"
  lang="en" xml:lang="en">
  <head
    profile="http://selenium-ide.openqa.org/profiles/test-case">
    <meta content="text/html; charset=UTF-8"
      http-equiv="Content-Type" />
    <link href="http://localhost:8080/" rel="selenium.base" />
    <title>Selenium test for SportController</title>
  </head>
  <body>
    <table border="1" cellpadding="1"
      cellspacing="1">
      <thead>
        <tr>
          <td colspan="3" rowspan="1">Selenium test for
            SportController</td>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>open</td>
          <td>http://localhost:8080/diaulos/sports?form</td>
          <td></td>
        </tr>
        <tr>
          <td>type</td><td>_code_id</td><td>someCode1</td>
        </tr>
        <tr>
          <td>type</td><td>_name_id</td><td>someName1</td>
        </tr>
        <tr>
          <td>type</td><td>_description_id</td>
          <td>someDescription1</td>
        </tr>
        <tr>
          <td>clickAndWait</td><td>//input[@id = 'proceed']</td>
          <td></td>
        </tr>
        <tr>
```

```
          <td>verifyText</td>
          <td>_s_com_atreceno_it_diaulos_domain_Sport_code_code_id
          </td>
          <td>someCode1</td>
        </tr>
        <tr>
          <td>verifyText</td>
          <td>_s_com_atreceno_it_diaulos_domain_Sport_name_name_id
          </td>
          <td>someName1</td>
        </tr>
        <tr>
          <td>verifyText</td>
          <td>_s_com_..._domain_Sport_description_description_id
          </td>
          <td>someDescription1</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

The code of the `Sport` entity was defined with 2 characters but Roo did not reflect that in the test case. The `test-sport.xhtml` file was managed by Roo, so any manual change would have been overwritten. To solve this issue there were two options: create custom test cases and replace them with the ones created by Roo in the file `test-suite.xhtml`; or create a separate folder with custom test cases and a separate test suite. To avoid mixing test cases, the second approach was taken. Additionally, for the creation of the new test cases, the tool Selenium IDE was used to record the actions of the user within the browser. The application Selenium IDE was downloaded from the Selenium website as a XPI (Cross-Platform Install) file, and then opened with Firefox. Once installed, the tool was accessed from the *Tools* menu in Firefox (see Figure 3.4).

The tests could be run from the Selenium IDE, but not from the command line (using Maven) because of a bug in Selenium with the current version of Firefox, 23.0.1 at the time of writing. The workaround was to downgrade Firefox to version 21 or to upgrade `selenium-server` to version 2.34. The latter option was not feasible because `selenium-maven-plugin` had a dependency on `selenium-server` version 2.25. Thankfully, there was a third alternative: To replace *Selenium Server* (formerly *Selenium RC*) with the improved *Selenium WebDriver*, which addressed some limitations in the Selenium RC API. The plugin `selenium-maven-plugin` was replaced in `pom.xml` by the new library as follows:

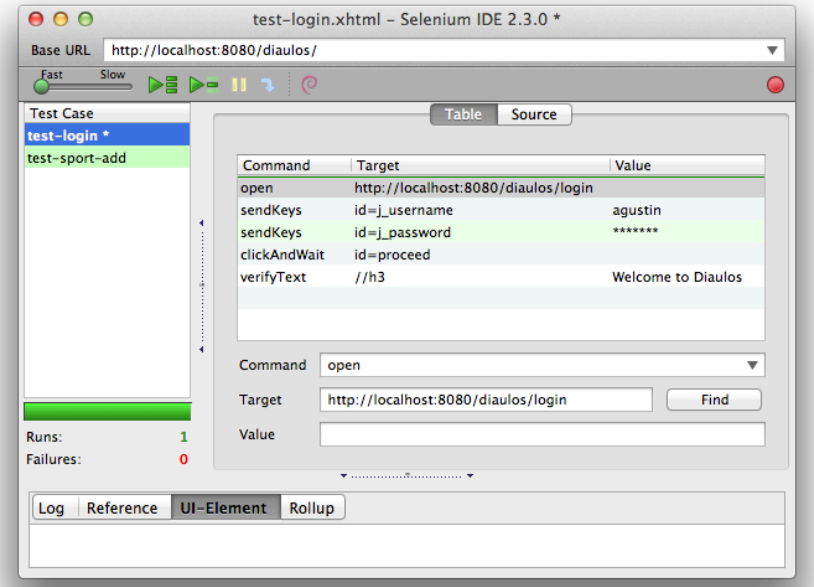

Figure 3.8: Selenium IDE

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>2.35.0</version>
</dependency>
```

Additionally, Selenium IDE allowed to export the HTML files to Java so they could be run from Eclipse IDE and even debugged if necessary. The new test cases were located in the `src/test/java` folder in the corresponding web package as recommended by Tahchiev and Massol (2011). As an example, the test case `SportTest.java` is shown in the next listing:

```
package com.atreceno.it.diaulos.web;

//import statements omitted

public class SportTest {
    private WebDriver driver;
    private String baseUrl;
    private boolean acceptNextAlert = true;
    private StringBuffer verificationErrors = new StringBuffer();

    @Before
    public void setUp() throws Exception {
        driver = new FirefoxDriver();
        baseUrl = "http://localhost:8080/diaulos/";
        driver.manage().timeouts().implicitlyWait(30,
            TimeUnit.SECONDS);
    }

    @Test
    public void testSport() throws Exception {

        driver.get(baseUrl);
        driver.findElement(By.linkText("Create new Sport"))
            .click();
        driver.findElement(By.id("_code_id")).sendKeys("AA");
        driver.findElement(By.id("_name_id"))
            .sendKeys("Selenium Test");
        driver.findElement(By.id("_description_id")).sendKeys(
                "This is a Selenium test.");
        driver.findElement(By.id("proceed")).click();
        try {
            assertEquals(
                "Selenium Test",
```

```
            driver.findElement(
                By.id("_s_..._domain_Sport_name_name_id"))
                .getText());
    } catch (Error e) {
        verificationErrors.append(e.toString());
    }
    driver.findElement(By.linkText("Find by Code Equals"))
        .click();
    driver.findElement(By.id("_code_id")).sendKeys("AA");
    driver.findElement(By.id("proceed")).click();
    try {
        assertEquals("This is a Selenium test.",
            driver.findElement(By
                .xpath("//table/tbody/tr[1]/td[2]"))
                .getText());
    } catch (Error e) {
        verificationErrors.append(e.toString());
    }
    driver.findElement(By.cssSelector("input.image")).click();
    assertTrue(closeAlertAndGetItsText().matches(
        "^Are you sure want to delete this item[\\s\\S]$"));
    driver.findElement(By.linkText("Find by Code Equals"))
        .click();
    driver.findElement(By.id("_code_id")).sendKeys("AA");
    driver.findElement(By.id("proceed")).click();
    try {
        assertEquals(
            "No Sport found.",
            driver.findElement(
                By.id("_title_..._domain_Sport_id_pane"))
                .getText());
    } catch (Error e) {
        verificationErrors.append(e.toString());
    }
}

@After
public void tearDown() throws Exception {
    driver.quit();
    String verificationErrorString = verificationErrors
        .toString();
    if (!"".equals(verificationErrorString)) {
        fail(verificationErrorString);
    }
}
```

```
    private String closeAlertAndGetItsText() {
        try {
            Alert alert = driver.switchTo().alert();
            String alertText = alert.getText();
            if (acceptNextAlert) {
                alert.accept();
            } else {
                alert.dismiss();
            }
            return alertText;
        } finally {
            acceptNextAlert = true;
        }
    }
}
```

To run the tests automatically, the plugin `maven-failsafe-plugin` was included in the file `pom.xml` as follows:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <version>2.16</version>
  <executions>
    <execution>
      <id>selenium-tests</id>
      <goals>
        <goal>integration-test</goal>
      </goals>
      <configuration>
        <includes>
          <include>**/SeleniumTestSuite*</include>
        </includes>
      </configuration>
    </execution>
    <execution>
      <id>verify</id>
      <goals>
        <goal>verify</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

This way the Selenium WebDriver tests were included in the `integration-test` goal (as it was done earlier with Selenium RC) and the tests could be run from

Selenium IDE, Eclipse IDE or from Maven, as seen below:

```
$ mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building diaulos 0.1.0.BUILD-SNAPSHOT
[INFO] ------------------------------------------------------------
[INFO]
[INFO] --- aspectj-maven-plugin:1.2:compile (default) @ diaulos -
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources @ diaulos ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 19 resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile @ diaulos ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- aspectj-maven-plugin:1.2:test-compile @ diaulos ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources @ diaulos ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory .../src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:testCompile @ diaulos ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12:test @ diaulos ---
[INFO] Surefire report directory: .../target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------

Results :

Tests run: 128, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 1:02.106s
[INFO] Finished at: Wed Sep 11 17:02:54 BST 2013
[INFO] Final Memory: 8M/81M
[INFO] ------------------------------------------------------------
```

To perform the tests, the database was loaded with over 600 events and 10,000 athletes from the London 2012 Olympic Games. The list of events was available online[7] in CSV format. The list of athletes was downloaded from the BBC website[8] with an HTTP parser using the library Apache HttpComponents and transforming the HTML to XML via XSLT (see Friesen 2011, chap. 10; Juneau 2013, chap. 22). When the application was loaded with the Olympic data, some performance issues were found and in order to analyse what the application was actually doing, a new instance of *Pivotal tc Server* with *Spring Insight* was installed. In addition, the queries to the database were monitored by turning on the global variable `general_log`:

```
$ ./bin/mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 280
Server version: 5.6.8-rc MySQL Community Server (GPL)

mysql> show variables like 'general%';
+------------------+-----------------------------------+
| Variable_name    | Value                             |
+------------------+-----------------------------------+
| general_log      | OFF                               |
| general_log_file | /usr/local/mysql/data/jupiter.log |
+------------------+-----------------------------------+
2 rows in set (0.00 sec)

mysql>
mysql> set global general_log = 1;
```

It was also possible to show the queries in the logs by setting the variable `hibernate.show_sql` to `true` in the `persistence.xml` configuration file, but it required restarting the server for the changes to take effect:

```
<property name="hibernate.show_sql" value="true"/>
<property name="hibernate.format_sql" value="true"/>
```

With the tool `mysqladmin`, it was possible to monitor the number of connections and the status of the server:

```
$ ./bin/mysqladmin -u root -p proc stat
Enter password:
+----+-----------+-----------+-----------+---------+------+
```

[7] http://odf.olympictech.org/
[8] http://www.bbc.co.uk/sport/olympics/2012/athletes

```
| Id | User      | Host      | db          | Command | Time |
+----+-----------+-----------+-------------+---------+------+
| 14 | root      | localhost |             | Query   | 0    |
| 11 | bricklane | localhost | diaulos_db | Sleep   | 8    |
+----+-----------+-----------+-------------+---------+------+
Uptime: 1351  Threads: 2  Questions: 64  Slow queries: 0
Opens: 88  Flush tables: 1  Open tables: 81
Queries per second avg: 0.047
```

Or even directly from the `mysql` client tool, a.k.a. monitor tool:

```
mysql> show processList;
+-----+-----------+-----------------+------------+---------+
| Id  | User      | Host            | db         | Command |
+-----+-----------+-----------------+------------+---------+
| 287 | bricklane | localhost:52632 | diaulos_db | Sleep   |
| 288 | root      | localhost:52634 | NULL       | Sleep   |
| 289 | root      | localhost:52635 | NULL       | Sleep   |
| 280 | root      | localhost       | diaulos_db | Query   |
+-----+-----------+-----------------+------------+---------+
4 row in set (0.00 sec)

mysql> show status;
+-------------------------------------------------+-------------+
| Variable_name                                   | Value       |
+-------------------------------------------------+-------------+
| Aborted_clients                                 | 16          |
| Aborted_connects                                | 7           |
| Binlog_cache_disk_use                           | 0           |
| Binlog_cache_use                                | 0           |
| Binlog_stmt_cache_disk_use                      | 0           |

  (some lines were omitted)

| Threads_cached                                  | 2           |
| Threads_connected                               | 2           |
| Threads_created                                 | 4           |
| Threads_running                                 | 1           |
| Uptime                                          | 529428      |
| Uptime_since_flush_status                       | 529428      |
+-------------------------------------------------+-------------+
339 rows in set (0.00 sec)

mysql>
```

Another useful tool to monitor the number of connections and the status of the server in real-time was MySQL Workbench, which was also used to generate the Enhanced Entity-Relationship (EER) diagrams. The tool was retrieving the information by querying the MySQL server with `show processList` every 6 seconds and `show global status` every 3 seconds.

3.4.4 Integration Tests

Using the same Selenium tests that were defined in the front-end unit tests, it was very easy to perform an integration test for the production environment. The only thing that needed to change was the base URL, now pointing to the application deployed to Heroku. The following logs show that it took about 40 seconds to perform all the tests successfully:

```
$ heroku logs --tail
2013-10-08T16:34:07.492500+00:00 heroku[router]: at=info
method=GET path=/login host=diaulos.herokuapp.com
fwd="86.160.254.197" dyno=web.1 connect=0ms service=35ms
status=200 bytes=4084
2013-10-08T16:34:07.745707+00:00 heroku[router]: at=info
method=POST path=/resources/j_spring_security_check
host=diaulos.herokuapp.com fwd="86.160.254.197" dyno=web.1
connect=0ms service=16ms status=302 bytes=0
2013-10-08T16:34:08.374581+00:00 heroku[router]: at=info
method=GET path=/ host=diaulos.herokuapp.com fwd="86.160.254.197"
dyno=web.1 connect=0ms service=588ms status=200 bytes=7968
2013-10-08T16:34:08.794045+00:00 heroku[router]: at=info
method=GET path=/ host=diaulos.herokuapp.com fwd="86.160.254.197"
dyno=web.1 connect=0ms service=54ms status=200 bytes=7967

(some lines omitted)

2013-10-08T16:34:46.077438+00:00 heroku[router]: at=info
method=POST path=/sports/AA host=diaulos.herokuapp.com
fwd="86.160.254.197" dyno=web.1 connect=0ms service=129ms
status=302 bytes=0
2013-10-08T16:34:46.185604+00:00 heroku[router]: at=info
method=GET path=/sports?page=1&size=10 host=diaulos.herokuapp.com
fwd="86.160.254.197" dyno=web.1 connect=0ms service=75ms
status=200 bytes=17429
2013-10-08T16:34:46.413424+00:00 heroku[router]: at=info
method=GET path=/sports?find=ByCodeEquals&form
host=diaulos.herokuapp.com fwd="86.160.254.197" dyno=web.1
connect=0ms service=21ms status=200 bytes=8880
2013-10-08T16:34:46.654914+00:00 heroku[router]: at=info
```

```
method=GET path=/sports?find=ByCodeEquals&code=AA
host=diaulos.herokuapp.com fwd="86.160.254.197" dyno=web.1
connect=0ms service=28ms status=200 bytes=7996
```

Figure 3.9 shows that the physical memory was very close to the limit of one web dyno, 512 MB.

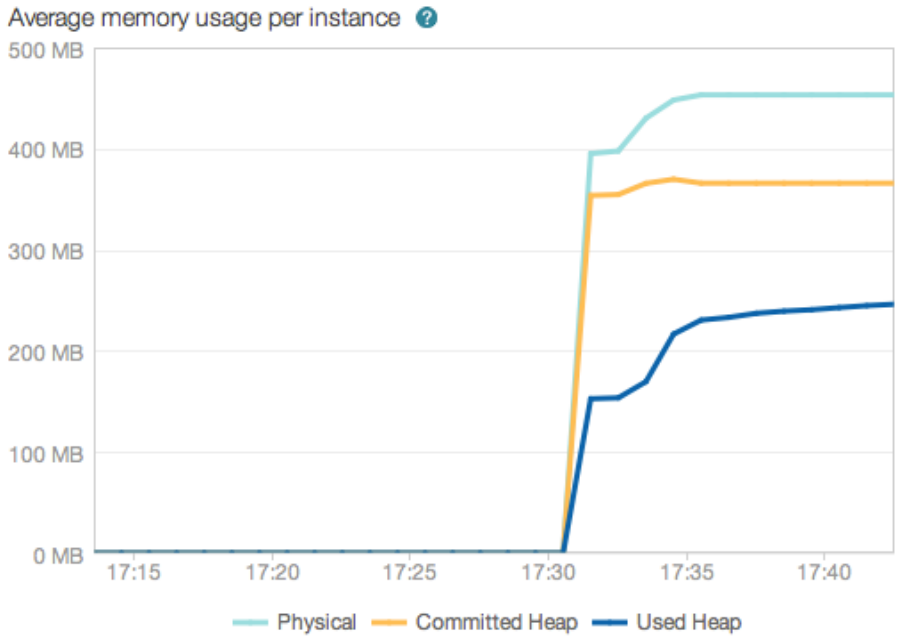

Figure 3.9: Memory usage with one web dyno

In fact, right after running the tests, the first Error R14 appeared in the logs:

```
$ heroku logs --tail
2013-10-08T16:37:20.922386+00:00 heroku[web.1]: Process running
mem=512M(100.0%)
2013-10-08T16:37:20.922527+00:00 heroku[web.1]: Error R14 (Memory
quota exceeded)
```

Resizing the web *dyno* to 1024 MB solved the problem:

```
$ heroku ps:resize web=2X
Resizing and restarting the specified dynos... done
web dynos now 2X ($0.10/dyno-hour)
```

As shown in the following logs, the application was restarted when the size of a *dyno* was changed:

```
$ heroku logs --source heroku
2013-10-08T17:14:03.144724+00:00 heroku[api]: Resize web to 2 by
agustin@gast.it.uc3m.es
2013-10-08T17:14:02.666174+00:00 heroku[web.1]: State changed
from up to starting
2013-10-08T17:14:04.466837+00:00 heroku[web.1]: Stopping all
processes with SIGTERM
2013-10-08T17:14:06.380845+00:00 heroku[web.1]: Process exited
with status 143
2013-10-08T17:14:20.496567+00:00 heroku[web.1]: Starting process
with command `java -Xmx384m -Xss512k -XX:+UseCompressedOops
-javaagent:target/dependency/newrelic-agent.jar
-Dspring.profiles.active=cloud
-jar target/dependency/jetty-runner.jar --port 53502
target/*.war`
2013-10-08T17:14:52.193989+00:00 heroku[web.1]: State changed
from starting to up
```

3.4.5 Non-functional Tests

There are several command line tools available to perform non-functional tests. Two of the most popular are `ab`, which stands for Apache Benchmark, and `siege`, developed by Jeff Fulmer. Both tools offer numerous options to customise tests such as the number of concurrent users, authentication, content-type, and so on. The main difference between the two is that `ab` is only for stress tests, while `siege` can be used for stress tests as well as load tests. Another popular alternative to the command line is Apache JMeter, which offers a Graphical User Interface and its scripts can be imported to BlazeMeter, a cloud testing provider. Selenium-Grid also allows to run tests on different machines in parallel, but it was not designed for load testing. Another great, easy to use tool for cloud testing is Blitz.

Typically, non-functional tests are split into three parts:

- Load testing, which implies testing the system under normal load.
- Stress testing, where the system is overloaded to its maximum capacity.
- Stability testing, which implies testing the system for a long period of time.

3.4.5.1 Load testing with Siege

Siege is an HTTP load testing and benchmarking utility written in C language.

In this case, the tool was installed in a remote server as follows:

```
$ wget http://www.joedog.org/pub/siege/siege-latest.tar.gz
$ tar -zxf siege-latest.tar.gz
$ cd siege-3.0.4
$ ./configure --prefix=${HOME}/usr --localstatedir=${HOME}/var \
--mandir=${HOME}/usr/share/man --with-ssl
$ make & make install
$ siege -V
SIEGE 3.0.4
```

The variable `HOME` was the home directory of a user without root privileges:

```
$ echo ${HOME}
/home/apt/agustin
```

The environment variables `PATH` and `MANPATH` were updated accordingly in the file `.bash_profile`:

```
if [ -d ~/usr/bin ] ; then
    PATH=~/usr/bin:"${PATH}"
fi

if [ -d ~/usr/share/man ]; then
    MANPATH=~/usr/share/man${MANPATH:-:}
    export MANPATH
fi
```

Because the application under test, *Diaulos*, had security enabled, `siege` had to authenticate before trying to access any content. This could be configured in the configuration file `${HOME}/.siegerc` as follows (all in one line):

```
login-url = http://diaulos.herokuapp.com/resources/j_spring_secur
ity_check POST j_username=borough&j_password=market
```

Before starting the first load test, a `GET` request was performed to validate the login URL:

```
$ siege -g -v http://diaulos.herokuapp.com/sports
POST /resources/j_spring_security_check  HTTP/1.0
Host: diaulos.herokuapp.com
Accept: */*
User-Agent: Mozilla/5.0 (pc-i686-linux-gnu) Siege/3.0.4
```

```
Connection: close
Content-type: application/x-www-form-urlencoded
Content-length: 36

j_username=borough&j_password=market

HTTP/1.1 400 BAD_REQUEST
Content-Length: 0
Connection: Close
```

The server complained of a bad request sent by the client. Sending the same request directly to port 80 showed the same error:

```
$ telnet diaulos.herokuapp.com 80
Trying 54.217.211.129...
Connected to elb000001-1978822061.eu-west-1.elb.amazonaws.com.
Escape character is '^]'.
POST /resources/j_spring_security_check  HTTP/1.0
Host: diaulos.herokuapp.com
Accept: */*
User-Agent: Mozilla/5.0 (pc-i686-linux-gnu) Siege/3.0.4
Connection: close
Content-type: application/x-www-form-urlencoded
Content-length: 36

j_username=borough&j_password=market

HTTP/1.1 400 BAD_REQUEST
Content-Length: 0
Connection: Close

Connection closed by foreign host.
```

Analysing the request in more detail, it was possible to see an extra space in the first line sent by the client. Unfortunately, this was not the only bug found in `siege`. This is what happened when trying a `GET` request in localhost:

```
$ siege -g "http://localhost:8080/diaulos/"
GET /diaulos/ HTTP/1.0
Host: localhost:8080
Accept: */*
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:26.0)
Connection: close

HTTP/1.1 302 Found
```

```
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=8C6BEB7104EA5837F4E2A1BA8624A60B; Path=/di
aulos/; HttpOnly
Location: http://localhost:8080/diaulos/login;jsessionid=8C6BEB71
04EA5837F4E2A1BA8624A60B
Content-Length: 0
Date: Wed, 09 Oct 2013 20:59:42 GMT
Connection: close

GET /http://localhost:8080/diaulos/login;jsessionid=8C6BEB7104EA5
837F4E2A1BA8624A60B HTTP/1.0
Host: localhost:8080
Cookie: JSESSIONID=8C6BEB7104EA5837F4E2A1BA8624A60B
Accept: */*
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:26.0)
Connection: close

HTTP/1.1 404 Not Found
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=utf-8
Content-Length: 1181
Date: Wed, 09 Oct 2013 20:59:42 GMT
Connection: close

<html><head><title>VMware vFabric tc Runtime 2.9.2.RELEASE/7.0.39
.B.RELEASE - Error report</title>
...(HTML content omitted)...
</body></html>
```

`siege` was misinterpreting the attribute `Location` and instead of requesting `/diaulos/login`, it was requesting `/http://localhost:8080/diaulos/login`, which did not exist. Thankfully, the source code was available and both issues were resolved with a few logs added to the code (visible with the option `--debug`). Additionally, a patch was created and applied to the instance installed in the server:

```
$ diff -Naur siege-3.0.4 siege-3.0.4-dev > siege-3.0.4.patch
$ scp siege-3.0.4.patch agustin@sochi.gast.it.uc3m.es:~/usr/src
$ ssh agustin@sochi.gast.it.uc3m.es
$ cd ~/usr/src/siege-3.0.4
$ patch -bp1 < ../siege-3.0.4.patch
patching file src/client.c
patching file src/http.c
$ make & make install
```

Once the patch was installed, `siege` could authenticate successfully and retrieved the data requested as shown below:

```
$ siege -g "http://diaulos.herokuapp.com/sports/CM"
POST /resources/j_spring_security_check HTTP/1.0
Host: diaulos.herokuapp.com
Accept: */*
User-Agent: Mozilla/5.0 (pc-i686-linux-gnu) Siege/3.0.4
Connection: close
Content-type: application/x-www-form-urlencoded
Content-length: 36

j_username=borough&j_password=market

HTTP/1.1 302 Found
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Location: http://diaulos.herokuapp.com/
Server: Jetty(8.y.z-SNAPSHOT)
Set-Cookie: JSESSIONID=160vu2709ncj1etf30ucjmne8;Path=/
Connection: Close

GET /sports/CM HTTP/1.0
Host: diaulos.herokuapp.com
Cookie: JSESSIONID=160vu2709ncj1etf30ucjmne8
Accept: */*
User-Agent: Mozilla/5.0 (pc-i686-linux-gnu) Siege/3.0.4
Connection: close

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Server: Jetty(8.y.z-SNAPSHOT)
Content-Length: 119
Connection: Close

{"code":"CM","description":"Say something cool about Cycling Moun
tain Bike","name":"Cycling Mountain Bike","version":0}
```

The first load test was run with the following command:

```
$ siege -c40 -d10 -t40M -q -i -f siege-urls.txt > siege.out 2>&1 &
```

It tested the application with 40 concurrent[9] customers making requests for 40 minutes. The requests were sent with a delay between 0 and 10 seconds,

[9]The amount of concurrent users was an estimation of the actual number of customers interested in receiving the feed.

generating a load of 480 RPM (Requests Per Minute). The `-q` parameter set the quiet mode so only the summary was printed. The `-i` parameter set the Internet mode, where URLs are picked up randomly from the file specified with the `-f` parameter. To allow the test to run independently in the server, the file descriptor `2` (standard error) was redirected to the file descriptor `1` (standard out) and this one to the file `siege.out`.

A few more tests were run with similar results. The summary report of one of these is shown below:

```
$ cat siege.out

Lifting the server siege...      done.

Transactions:              18578 hits
Availability:             100.00 %
Elapsed time:            2400.03 secs
Data transferred:           9.55 MB
Response time:              0.12 secs
Transaction rate:           7.74 trans/sec
Throughput:                 0.00 MB/sec
Concurrency:                0.96
Successful transactions:   18578
Failed transactions:           0
Longest transaction:        0.80
Shortest transaction:       0.10
```

The number of `transactions` or hits was slightly lower than 480 hits per minute multiplied by 40 minutes (19200 hits) because the average time between requests was slightly bigger than 5 seconds. The `elapsed time` was the duration of the test, 40 minutes, as specified by the `-t` parameter. The `data transferred` was the amount of data transferred from the server, including the header information. The `response time` was calculated as the `elapsed time` divided by the number of transactions. `Transaction rate` was the average number of transactions the server was able to handle per second. `Throughput` was the data transferred divided by the elapsed time, which comes to 0.00398 MB/sec. `Concurrency` is defined vaguely as the average number of simultaneous connections but, looking at the source code, it was actually calculated as the `response time` multiplied by the `transaction rate`. The rest of the parameters are self-explanatory.

The previous tests corresponded to 40 concurrent users accessing the REST interface to get information from the competition. But, what if the competition is running at the same time… To simulate the competition, the worst case scenario was considered. During the London 2012 Olympic Games, the sporting event that could generate the most traffic was the Road Cycling Men (`CRM012`) when all 139 participants were crossing the intermediate point at the same time:

```
mysql> select pe.event_id, e.code, e.name,
count(pe.participant_id) as num from partic_event pe inner join
event e where pe.event_id = e.id group by pe.event_id order by
num desc limit 15;
+----------+--------+------------------+-----+
| event_id | code   | name             | num |
+----------+--------+------------------+-----+
|      205 | FBM400 | Men              | 351 |
|      207 | FBW400 | Women            | 265 |
|      309 | HOM400 | Men              | 216 |
|      311 | HOW400 | Women            | 216 |
|      303 | HBM400 | Men              | 178 |
|      305 | HBW400 | Women            | 172 |
|      615 | WPM400 | Men              | 156 |
|      560 | VOM400 | Men              | 144 |
|      562 | VOW400 | Women            | 144 |
|       87 | BKM400 | Men              | 144 |
|       89 | BKW400 | Women            | 143 |
|      150 | CRM012 | Men's Road Race  | 139 |
|       69 | ATW099 | Women's Marathon | 118 |
|       45 | ATM099 | Men's Marathon   | 105 |
|      617 | WPW400 | Women            | 104 |
+----------+--------+------------------+-----+
15 rows in set (0.01 sec)
```

The closest intermediate points were 15 Km apart and it took cyclists about 20 minutes to complete each segment. The Women's marathon event (ATW099) had a similar load with 118 participants and intermediate points every 5 Km, which it took runners about 15 minutes to complete. Therefore, a good estimation of the load of the sporting event with the most traffic would be 139 results simultaneously, every 15 minutes. However, during the London 2012 Games there were several competitions running at the same time. One of the busiest times was July 30, 2012 at 4:00pm with 18 competitions generating results simoultaneously. With `siege` it was possible to generate such a load by sending 139 results with a delay between 0 and 60 seconds, i.e, an average of 30 high load competitions running at the same time:

```
$ siege -c1 -d60 -t40M -q "http://diaulos.herokuapp.com/particlaps
/jsonArray POST < laps.json" > siege-c1-d60-t40M-q-run3.out 2>&1
```

The file `laps.json` contained 139 results similar to the following:

```
[
  {
    "lap":{"id":1,"version":0},
```

```
    "participant":{"id":213  ,"version":0},
    "rank":  1,
    "result":"100",
    "version":0
  },
  {
    "lap":{"id":1,"version":0},
    "participant":{"id":259  ,"version":0},
    "rank":  2,
    "result":"100",
    "version":0
  },
  ...
]
```

To compare how the application responded to different input loads, two different tests of 40 minutes each were performed:

- Load Test A: Simulating 40 concurrent customers using the REST interface every 5 seconds while the competitions are not running.
- Load Test B: Same as Test A but with 30 competitions running simultaneously.

The tests were scheduled using the `cron` utility as follows:

```
$ crontab -l
# m h dom mon dow command
0 * * * * /home/apt/agustin/simulate-users.sh
0 */2 * * * /home/apt/agustin/simulate-competitions.sh
```

Both scripts were just wrappers of the tool `siege` as shown below:

```
$ cat /home/apt/agustin/simulate-users.sh
/home/apt/agustin/usr/bin/siege -c40 -d10 -t40M -q -i -f \
siege-urls.txt >> siege-c40-d10-t40M.out 2>&1
$ cat /home/apt/agustin/simulate-competitions.sh
/home/apt/agustin/usr/bin/siege -c1 -d60 -t40M -q \
"diaulos.herokuapp.com/particlaps/jsonArray POST < laps.json"
>> siege-c1-d60-t40M.out 2>&1
```

This way, the load tests A and B were alternating every hour. Figure 3.10 shows the average server response time by tier for three hours (tests A - B - A). Runs 1 and 3 show similar response times but not exactly the same because `siege` simulated users by randomly hitting the URLs specified in the file `siege-urls.txt`

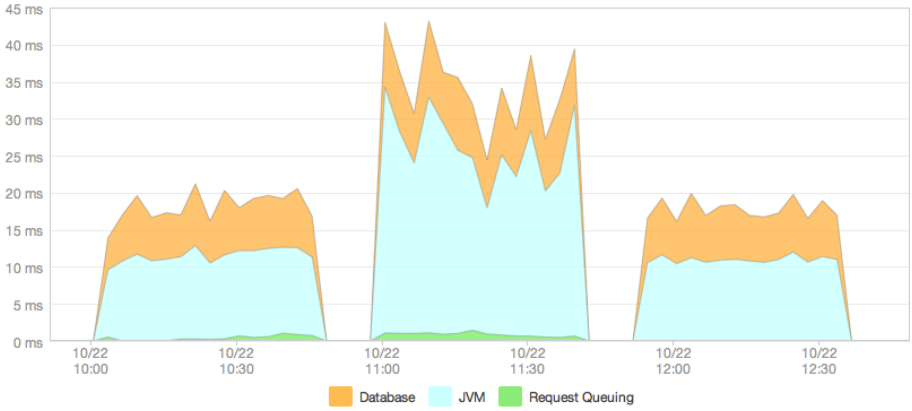

Figure 3.10: Server Response Time - Load Testing - Diaulos

and also because the delay between requests was a random interval between 0 and the number of seconds specified in the parameter `-d`.

The average server response time was 18 ms. for the first run, 34 ms. for the second run and 17 ms. for the third. The *request queuing* was the time elapsed since the request was received by the router until the request reached the web application. The summary report generated by `siege` when running `simulate-users.sh` is shown in Table 3.2. It is worth to mention the longest transaction of the second run as a result of the traffic generated by the competition. The summary report generated by `siege` when running `simulate-competitions.sh` is shown in Table 3.3.

Siege Statistics	Run 1	Run 2	Run 3
Transactions	18709	18647	18707
Availability (%)	100.00	100.00	100.00
Elapsed time (sec)	2399.70	2399.91	2400.14
Data transferred (MB)	9.60	9.56	9.65
Response time (sec)	0.13	0.13	0.13
Transaction rate (1/sec)	7.80	7.77	7.79
Throughput (KB/sec)	4.00	3.98	4.02
Concurrency	1.00	1.00	0.98
Successful transactions	18709	18647	18707
Failed transactions	0	0	0

Longest transaction (sec)	1.58	5.47	1.68
Shortest transaction (sec)	0.10	0.10	0.10

Table 3.2: User Simulation - Load Testing - Diaulos

Siege Statistics	Run 1	Run 2	Run 3
Transactions	N/A	73	N/A
Availability (%)	N/A	100.00	N/A
Elapsed time (sec)	N/A	2399.29	N/A
Data transferred (MB)	N/A	0.00	N/A
Response time (sec)	N/A	3.92	N/A
Transaction rate (1/sec)	N/A	0.03	N/A
Throughput (KB/sec)	N/A	0.00	N/A
Concurrency	N/A	0.12	N/A
Successful transactions	N/A	73	N/A
Failed transactions	N/A	0	N/A
Longest transaction (sec)	N/A	5.56	N/A
Shortest transaction (sec)	N/A	0.15	N/A

Table 3.3: Competition Simulation - Load Testing - Diaulos

The number of transactions were, as expected, slightly lower than 80, which added to the 18647 requests from the simulation of users, resulting in about the same request per minute in all three runs (see Figure 3.11). The data transferred was truncated to zero because the server responded with headers such as the following:

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
Server: Jetty(8.y.z-SNAPSHOT)
Connection: Close
```

This header was 114 Bytes, which multiplied by 80 requests, comes to less than 9 KB.

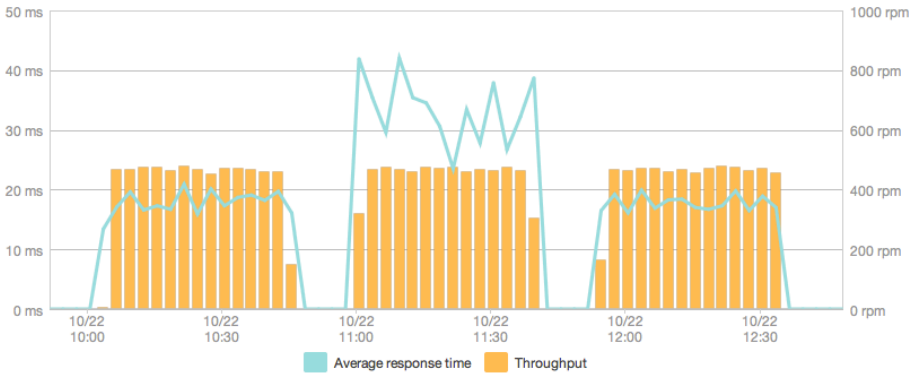

Figure 3.11: Server Throughput - Load Testing - Diaulos

The overall response of the application was excellent, with an Apdex score above 0.94 (see Figure 3.12). The Apdex score is defined as follows:

> The Apdex score is a ratio value of the number of satisfied and tolerating requests to the total requests made. Each satisfied request counts as one request, while each tolerating request counts as half a satisfied request.

This ratio is commonly used as a Key Performance Indicator (KPI) of how satisfied users are. In this case, the application's Apdex T-value was set to 0.1 seconds. This means requests responding in less than 0.1 seconds are satisfying, between 0.1 and 0.4 seconds are tolerating, and responding in more than 0.4 seconds are frustrating.

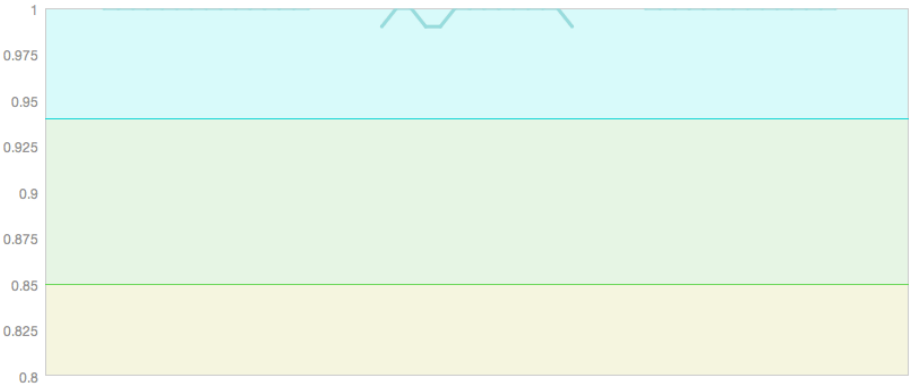

Figure 3.12: Apdex Score - Load Testing - Diaulos

Figure 3.13 shows the most time consuming web transactions calculated as the

average response time multiplied by the number of transactions in that period. The complete list of web transactions with percentages is shown in Figure 3.14. And a more detailed calculation of those numbers is shown in Table 3.4.

Web tx.	Resp.(ms)	Req./min.	Requests	Time(ms)	%
1	3860	0.4	72	277.92	21.45
2	26.6	28.0	5040	134.064	10.35
3	24.3	25.5	4590	111.537	8.61
4	23.2	26.0	4680	108.576	8.38
5	21.8	25.9	4662	101.6316	7.85
6	19.9	25.6	4608	91.6992	7.08
7	19.5	25.5	4590	89.505	6.91
8	16.1	25.7	4626	74.4786	5.75
9	15.0	26.7	4806	72.09	5.57
10	14.7	25.6	4608	67.7376	5.23
11	12.1	26.0	4680	56.628	4.37
12	11.9	25.9	4662	55.4778	4.28
13	11.9	24.4	4392	52.2648	4.03
14	14.9	0.672	120.96	1.802304	0.14

Table 3.4: Web Transactions - Load Testing - Diaulos

The first column of the table corresponds to the web transactions shown in Figure 3.14, in order of occurrence. The most time consuming web transaction was `ParticLapController/createFromJsonArray`, used to send the results of the competition. It was followed by the more complex entities with `1:N` or `N:1` relationships.

Figure 3.15 shows the database response time. The blue line corresponds to the simulation of the competition.

The database throughput, measured in calls per minute, is shown in Figure 3.16. It was around 1500 calls per minute and more than 480 requests per minute since one web transaction involves several calls to the database.

Figure 3.17 shows the most time consuming database operation was a `select` operation over the table `event`. This was not because of a bad query execution plan but because the table `event` was accessed by three different callers:

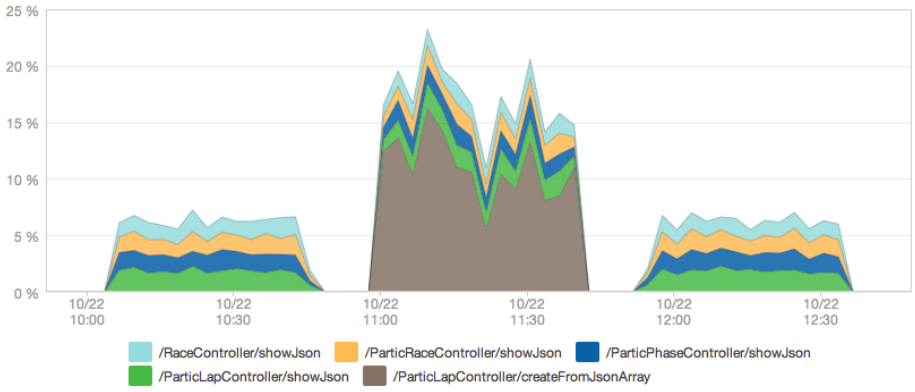

Figure 3.13: Top Web Transactions - Load Testing - Diaulos

/ParticLapController/createFromJsonArray	21.4%
/ParticLapController/showJson	10.3%
/ParticPhaseController/showJson	8.61%
/ParticRaceController/showJson	8.37%
/RaceController/showJson	7.84%
/ParticEventController/showJson	7.09%
/LapController/showJson	6.91%
/PhaseController/showJson	5.76%
/ParticipantController/showJson	5.58%
/EventController/showJson	5.24%
/VenueController/showJson	4.36%
/CountryController/showJson	4.28%
/SportController/showJson	4.04%
/CharacterEncodingFilter	0.14%

Figure 3.14: List of Web Transactions - Load Testing - Diaulos

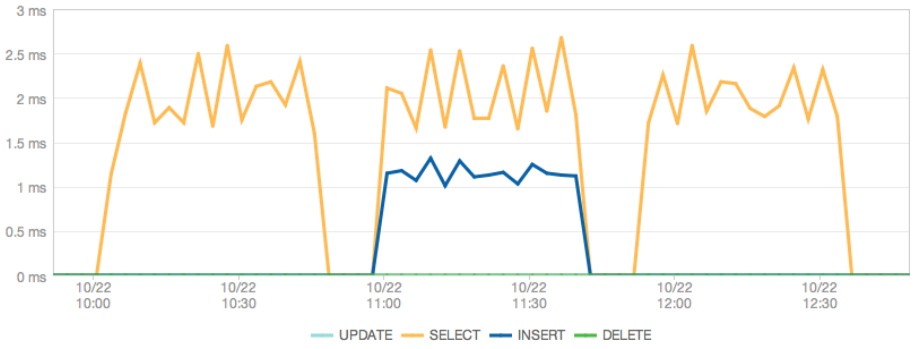

Figure 3.15: Database Response Time - Load Testing - Diaulos

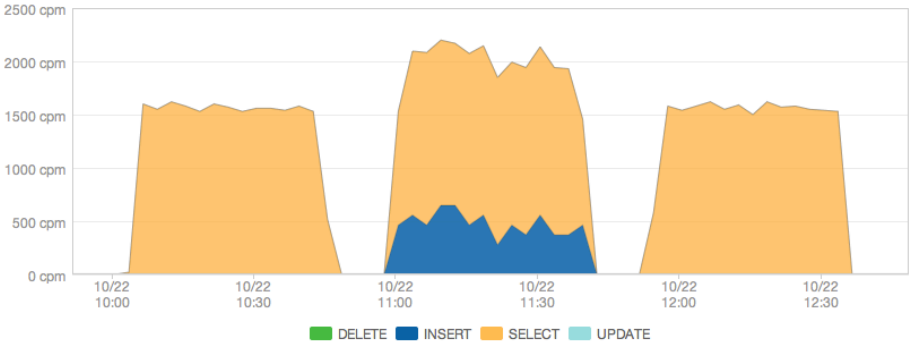

Figure 3.16: Database Throughput - Load Testing - Diaulos

- `/EventController/showJson`
- `/LapController/showJson`
- `/ParticRaceController/showJson`

The next most time consuming database operation was again a `select` operation, this time over the table `phase`, which was called from two different URLs:

- `/ParticLapController/showJson`
- `/PhaseController/showJson`

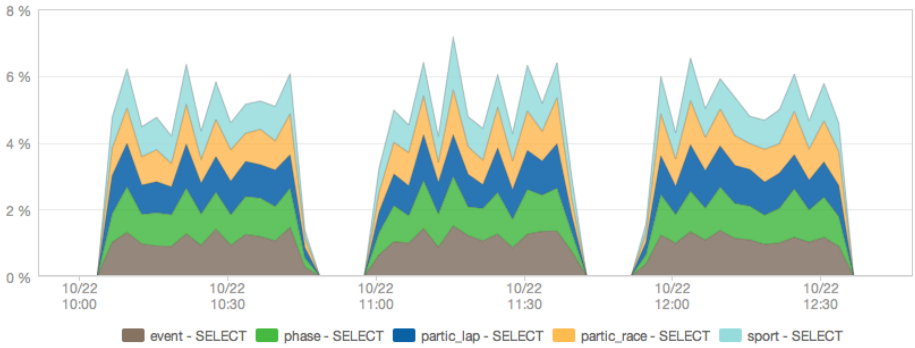

Figure 3.17: Top DB Operations - Load Testing - Diaulos

The number of instances and load of the Heroku web *dyno* is shown in Figure 3.18 and the CPU usage in Figure 3.19. The physical memory usage was below 350 MB, as shown in Figure 3.20. The garbage collector usage is shown in Figure 3.21.

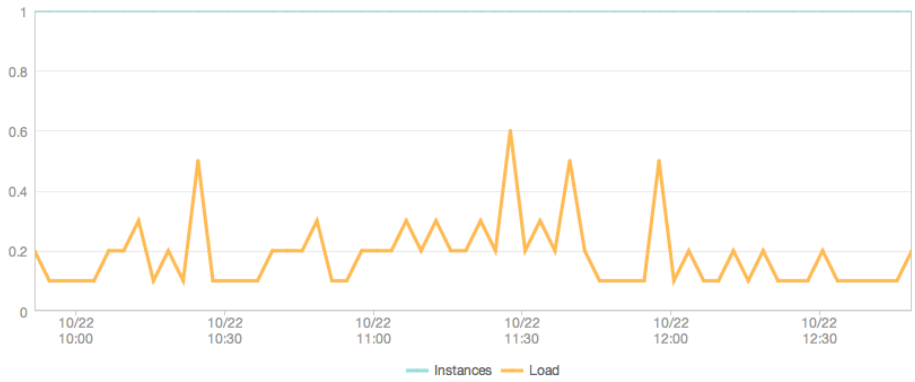

Figure 3.18: Number of instances and Load - Load Testing - Diaulos

Figure 3.19: CPU Usage - Load Testing - Diaulos

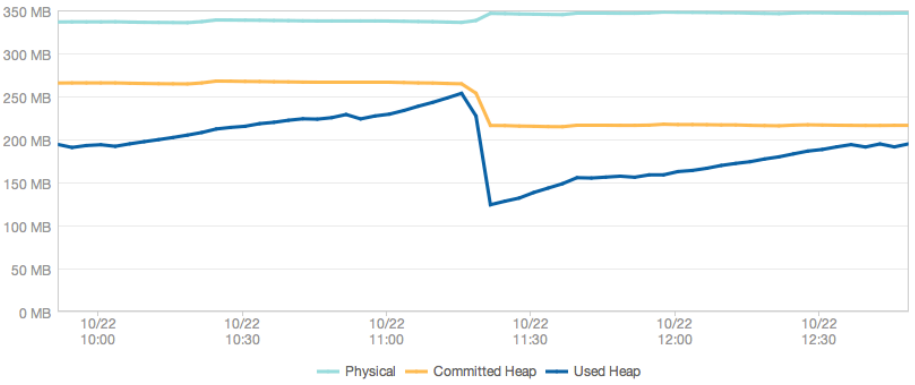

Figure 3.20: Memory Usage - Load Testing - Diaulos

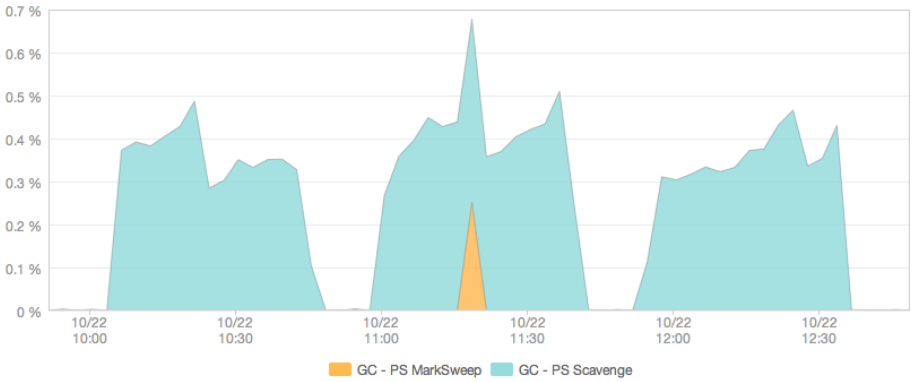

Figure 3.21: Garbage Collector Usage - Load Testing - Diaulos

3.4.5.2 Stress Testing with Siege

During a load test, the application is tested under a normal conditions. A stress test goes one step further and it pushes the application to the limits. Three runs were scheduled to simulate 40, 60 and 80 concurrent customers. All tests were using the REST interface without any delay between requests (benchmark mode) and 30 competitions running simultaneusly.

The summary report generated by `siege` for the simulation of users is shown in Table 3.5. In all three runs, the application was able to process more than half a million requests in 40 minutes, giving a rate of 250 requests per second, 30 times bigger than the rate for the load test performed in the previous section. The data transferred is also bigger as expected. The main difference between the tests is the longest transaction time, reaching a value of 94.47 seconds when simulating 80 concurrent customers. This is because the transaction was waiting to get a free connection from the pool.

Table 3.6 shows the summary report corresponding to the simulation of the competitions. The load was the same throughout the tests, 30 simultaneous competitions sending 139 results every 15 minutes. Again, the main difference is in the longest transaction time, 8.05, 11.98 and 24.33 seconds for 40, 60 and 80 concurrent customers, respectively.

Siege Statistics	Run 1	Run 2	Run 3
Transactions	549688	608425	600261
Availability (%)	100.00	100.00	100.00
Elapsed time (sec)	2399.55	2399.79	2400.88
Data transferred (MB)	282.88	313.03	308.69
Response time (sec)	0.17	0.24	0.32
Transaction rate (1/sec)	229.08	253.53	250.02
Throughput (MB/sec)	0.12	0.13	0.13
Concurrency	39.83	59.59	79.42
Successful transactions	549688	608425	600261
Failed transactions	0	1	0
Longest transaction (sec)	5.66	93.24	94.47
Shortest transaction (sec)	0.10	0.10	0.10

Table 3.5: User Simulation - Stress Testing - Diaulos

Siege Statistics	Run 1	Run 2	Run 3
Transactions	69	69	68
Availability (%)	100.00	100.00	100.00
Elapsed time (sec)	2399.64	2399.42	2399.71
Data transferred (MB)	0.00	0.00	0.00
Response time (sec)	4.06	4.62	5.56
Transaction rate (1/sec)	0.03	0.03	0.03
Throughput (MB/sec)	0.00	0.00	0.00
Concurrency	0.12	0.13	0.16
Successful transactions	69	69	68
Failed transactions	0	0	0
Longest transaction (sec)	8.05	11.98	24.33
Shortest transaction (sec)	0.15	0.33	0.36

Table 3.6: Competition Simulation - Stress Testing - Diaulos

The server response time is shown in Figure 3.22. During load testing there was an average response time of 34 ms. with the competition running. In benchmarking mode, the response time was around 45 ms, which is quite acceptable. There was a considerable increase in the request queuing time because of the greater number of requests per second.

The server throughput is shown in Figure 3.23 and it corroborates the result given by `siege` with 15,000 requests per minute or 250 requests per second.

Interestingly, the more load the application had, the better the Apdex score, as shown in Figure 3.24. This was because even though there were more frustrated requests, the number of successful requests was even higher.

The most time consuming web transactions are shown in Figure 3.25. The main difference with the results shown during load testing is that `/ParticLapController/createFromJsonArray` was no longer the most time consuming. This was due to the considerable increase of the number of requests of all the other web transactions. However, the rest of the transactions were in the same order.

The average database response time, shown in Figure 3.26, was increased from 2 ms. to 3 ms. for `select` operations, and from 1 ms. to 3 ms. for `insert`

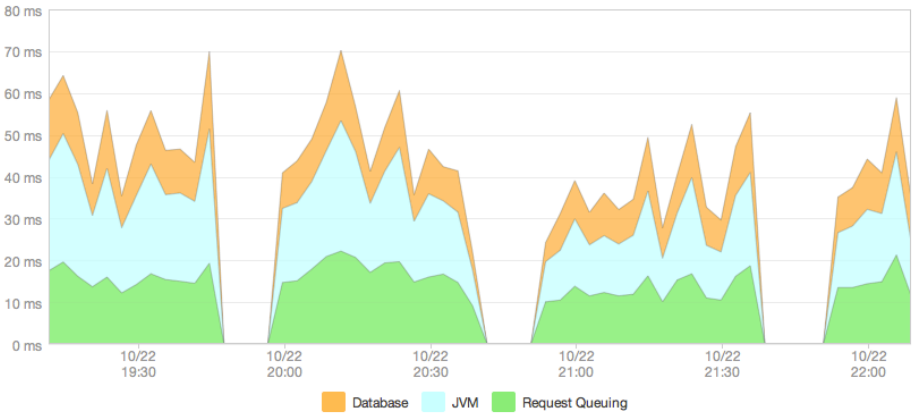

Figure 3.22: Server Response Time - Stress Testing - Diaulos

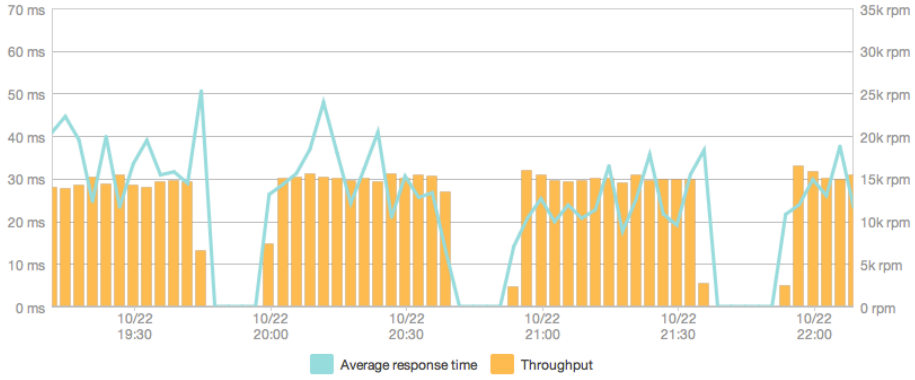

Figure 3.23: Server Throughput - Stress Testing - Diaulos

Figure 3.24: Apdex Score - Stress Testing - Diaulos

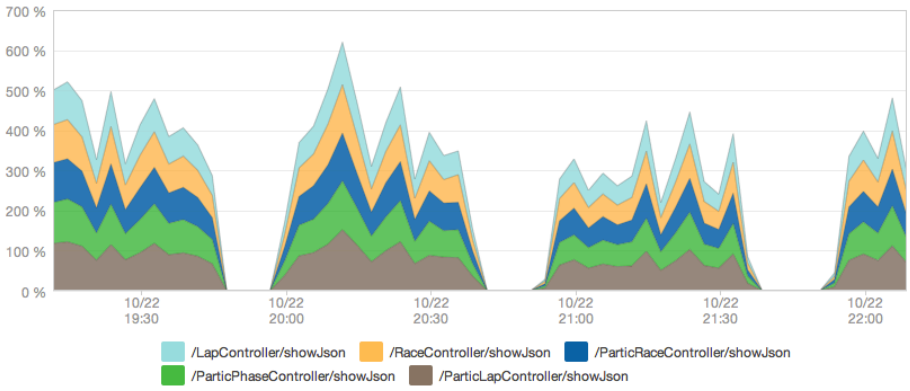

Figure 3.25: Top Web Transactions - Stress Testing - Diaulos

operations. However, the standard deviation was bigger this time because there were not enough connexions available, as shown in the server logs:

```
$ heroku logs --tail
2013-10-22T21:16:27.401698+00:00 app[web.1]: com.mysql.jdbc.
exceptions.jdbc4.MySQLNonTransientConnectionException: Too many
connections
```

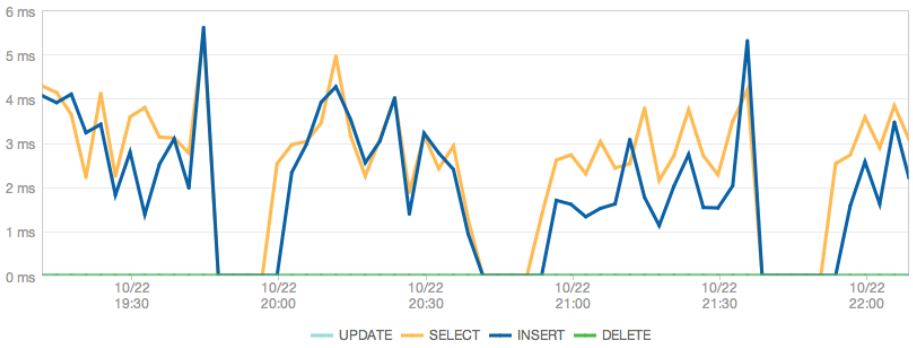

Figure 3.26: Database Response Time - Stress Testing - Diaulos

The database throughput reached 50,000 calls per minute, as shown in Figure 3.27. This was mostly due to `select` operations since the `insert` operations remained the same.

The most time consuming database operations by wall-clock time are shown in Figure 3.27. The order was not exactly the same as during the load testing because of the random nature of the tool `siege`.

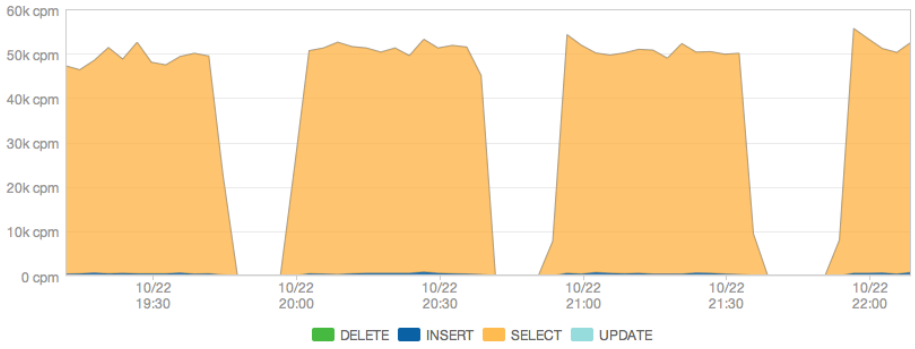

Figure 3.27: Database Throughput - Stress Testing - Diaulos

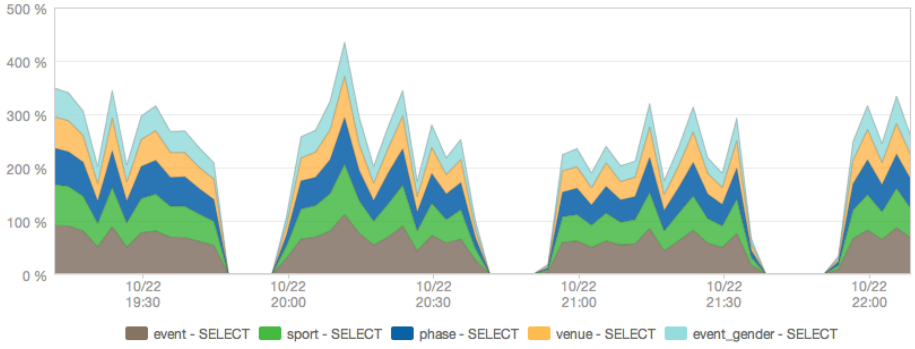

Figure 3.28: Top DB Operations - Stress Testing - Diaulos

The number of instances and load are shown in Figure 3.29. The CPU usage (shown in Figure 3.30) was near its maximum capacity. The heap memory was still around 200 MB (see Figure 3.31) because there was also more garbage collector activity as shown in Figure 3.32.

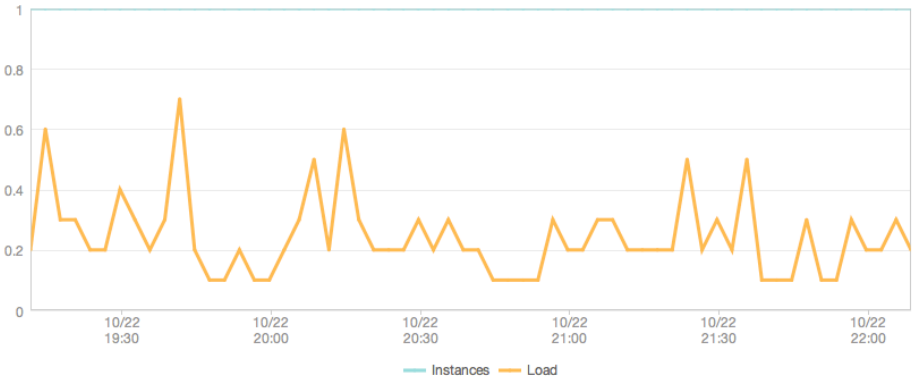

Figure 3.29: Number of instances and Load - Stress Testing - Diaulos

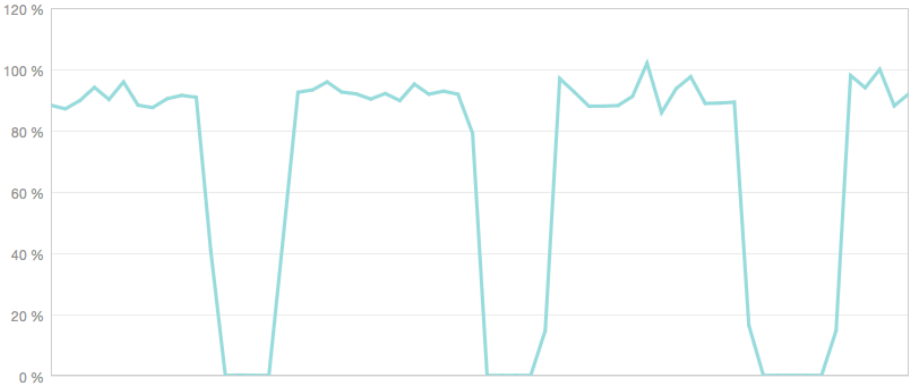

Figure 3.30: CPU Usage - Stress Testing - Diaulos

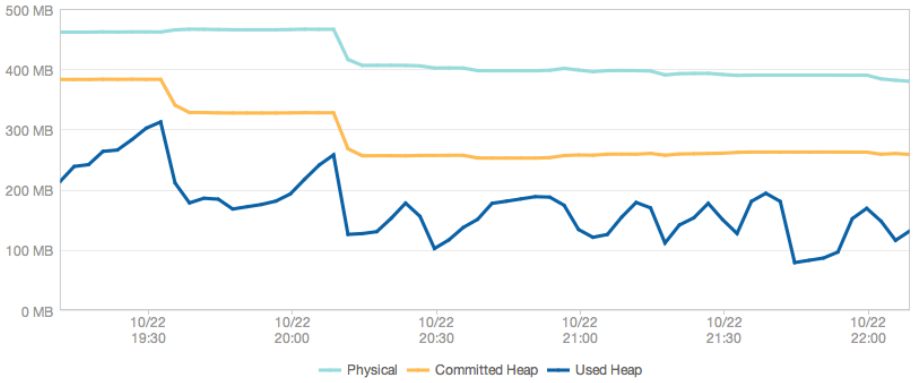

Figure 3.31: Memory Usage - Stress Testing - Diaulos

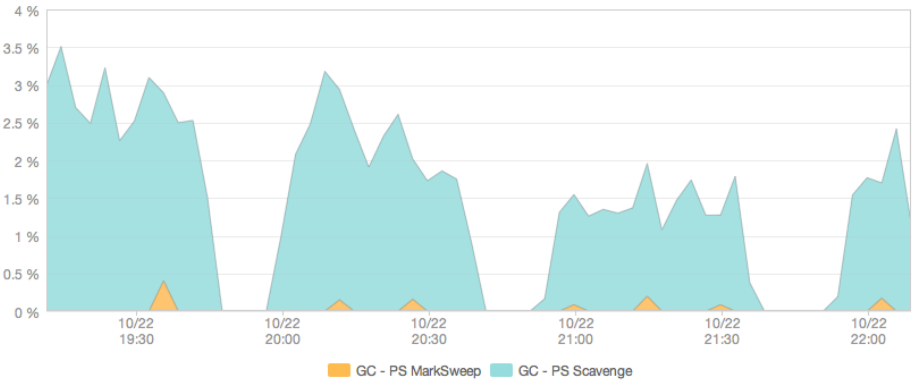

Figure 3.32: Garbage Collector Usage - Stress Testing - Diaulos

4 NodeJS Solution Architecture

Success is 1% inspiration, 98% perspiration, and 2% attention to detail.
—Anonymous

Chapter 4 covers the second solution architecture, called *Stadion*. It is also a web-based application, however this time it uses JavaScript in both the front and the back ends. Section 4.1 gives a brief introduction to NodeJS, which allows for writing JavaScript applications on the server-side. Section 4.2 focuses on the implementation, including the scaffolding of the application, the responsive design and the building process. Then, section 4.3 explains how to deploy the application in the cloud. Finally, section 4.4 covers the unit tests, end-to-end tests and non-functional tests.

4.1 Introduction to NodeJS

JavaScript is one of the major implementations of ECMAScript, a scripting language defined in the ECMA-262 specification by Ecma International. JavaScript is becoming very popular. In fact, at the time of writing, it is the most popular programming language on GitHub, followed by Ruby and Java (see Figure 4.1).

This can be attributed to two factors: the race to develop even faster JavaScript Engines, and the increased popularity of NodeJS, a platform for writing JavaScript applications on the server-side. Server-side JavaScript is not a new concept and it was announced shortly after Netscape released LiveScript for browsers in 1995. NodeJS runs on V8, a JavaScript engine written in C++ developed by Google. Even though ECMA-262 is the official JavaScript specification, it is focused on applications running in a browser. CommonJS

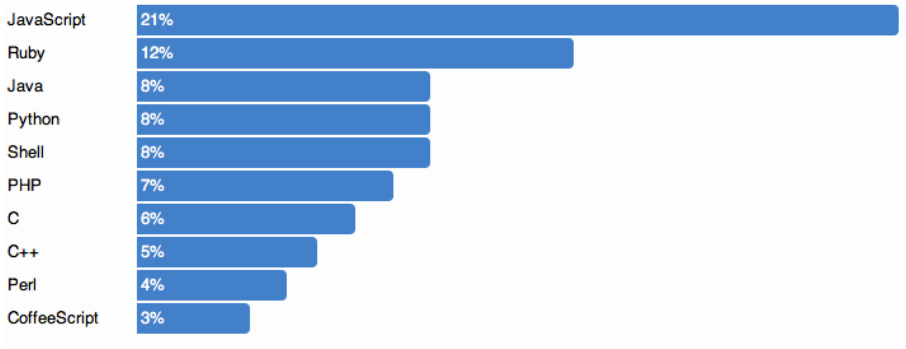

Figure 4.1: Programming languages trend (source: GitHub)

tries to fill that gap by specifying an ecosystem for JavaScript applications outside the browser.

For more information about JavaScript, Flanagan (2012), Flanagan and Loukides (2011) and Resig and Bibeault (2013) are three excellent references. For a more in-depth view of NodeJS, Wilson (2012) and Powers (2012) introduce this platform from a practice point of view.

4.2 Design and Implementation

4.2.1 JavaScript Scaffolding

When writing an application, it is important to use the right tools to help improve productivity, while using good practices at the same time. Maven is a good example of this, becoming the *de facto* build and package management tool in the Java world. The counterpart in the JavaScript World is Yeoman, a collection of tools and best practices to manage JavaScript applications. Yeoman consists of three components:

- Yo, to scaffold out a new application.
- Grunt, to build, preview and launch the tests.
- Bower, to manage package dependency.

NodeJS comes with a package manager, Node Packaged Modules (NPM), to install these and other components. There are several ways of installing NodeJS. Using the command line is quite universal and it is the method shown below:

```
$ curl -L http://nodejs.org/dist/latest/node-v0.10.6.pkg > \
node-v0.10.6.pkg
```

```
$ shasum node-v0.10.6.pkg
cd0acf9b332c30aba6a72979d3373e342fad6b95  node-v0.10.6.pkg
$ curl -s http://nodejs.org/dist/latest/SHASUMS.txt | grep \
"node-v0.10.6.pkg"
cd0acf9b332c30aba6a72979d3373e342fad6b95  node-v0.10.6.pkg
$ sudo installer -pkg node-v0.10.6.pkg -target /
$ node --version
v0.10.6
$ npm --version
1.2.18
```

Once installed, it was straithforward to install Yeoman's components:

```
$ npm install -g yo grunt-cli bower
```

Yo provided several generators to create the scaffolding of a JavaScript application. Because the application implemented was based on AngularJS, the generator-angular was used as follows:

```
$ npm search yeoman-generator
$ npm install -g generator-angular
$ mkdir stadion && cd $_
$ yo angular --skip-install
$ bower install
$ npm install
$ git init
$ git add .
$ git commit -m "Create scaffolding of the project"
```

The initial file structure was the following:

```
$ tree . --charset=ASCII -L 2
.
|-- Gruntfile.js
|-- README.md
|-- app
|   |-- 404.html
|   |-- favicon.ico
|   |-- robots.txt
|   |-- scripts
|   |-- styles
|   `-- views
|-- bower.json
|-- karma-e2e.conf.js
```

```
|-- karma.conf.js
|-- package.json
`-- test
    |-- runner.html
    `-- spec
```

The project dependencies were managed by two different files: `package.json` (used by `npm`) and `bower.json` (used by `bower`). Sometimes, both package managers are mixed up but in summary `npm` is used as a package manager for NodeJS applications (back-end) and `bower` is used as a package manager for the browser (front-end).

Even though there was an option to add Twitter Bootstrap when creating an application with `yo`, it was installed with `bower` because `yo` actually used compass-twitter-bootstrap, which is usually a version or two behind the latest release. grunt-contrib-less was also installed to compile LESS files into CSS:

```
$ bower install bootstrap --save
$ npm install grunt-contrib-less --save-dev
```

The `--save` and `--save-dev` parameters were used to add an entry to the `dependencies` and `devDependences`, respectively. It is also worth mentioning the importance of following best practices when defining the dependencies in the `package.json` file. For more information, there is an excellent article online[1] entitled *Package Dependencies Done Right*.

Once the scaffolding for the front-end part of the project was created, the server counterpart could be added. At the time of writing, a best practice to create an application with JavaScript in the front-end and the back-end simultaneously, did not exist. Some authors preferred to distinguish between the client and the server by creating two different folders. This was the approach taken by Meteor, an open-source platform for building JavaScript applications. Brian Ford, who worked on AngularJS at Google, had a different approach and created an angular-express-seed to address this issue. Unfortunately, this seed had some limitations. Firstly, its dependencies were not up to date due to a lack of a package manager. Secondly, it kept all the front-end files in the `public` directory, which was adequate for small projects but not for scalable projects using other technologies such as Jade or LESS.

The final folder structure of the project was influenced by Express, a popular web application framework for NodeJS, as shown below:

```
$ npm view express version
3.4.4
$ sudo npm install --global express
```

[1] http://blog.nodejitsu.com/package-dependencies-done-right

```
/usr/local/bin/express ->
/usr/local/lib/node_modules/express/bin/express
express@3.4.4 /usr/local/lib/node_modules/express
|-- methods@0.1.0
|-- cookie-signature@1.0.1
|-- range-parser@0.0.4
|-- fresh@0.2.0
|-- debug@0.7.3
|-- buffer-crc32@0.2.1
|-- cookie@0.1.0
|-- mkdirp@0.3.5
|-- commander@1.3.2
|-- send@0.1.4
`-- connect@2.11.0
$ express --version
3.4.4
$ mkdir stadion-express; cd $_
$ express --css less
$ tree stadion-express --charset=ASCII -L 3
stadion-express
|-- app.js
|-- package.json
|-- public
|   |-- images
|   |-- javascripts
|   `-- stylesheets
|       `-- style.less
|-- routes
|   |-- index.js
|   `-- user.js
`-- views
    |-- index.jade
    `-- layout.jade

6 directories, 7 files
```

The next section analyses two of the most popular frameworks to make the application responsive.

4.2.2 Responsive Web Design

The best way to make the application *Stadion* responsive was to use either Zurb Foundation or Twitter Bootstrap. To help with the decision of which one to use, both frameworks were analysed.

4.2.2.1 Zurb Foundation

Zurb Foundation offered several ways to build a web project:

- Plain CSS (default or customised).
- Compass Gem using Sass/SCSS.
- Using Bower.

The first way did not require Sass or any other tool because it only contained vanilla CSS. It was possible to get a ZIP file from the Foundation download page[2]. Once the file was extracted it contained the following file structure:

```
$ tree foundation-4 --charset=ASCII -L 3
foundation-4
|-- css
|   |-- foundation.css
|   |-- foundation.min.css
|   `-- normalize.css
|-- humans.txt
|-- img
|-- index.html
|-- js
|   |-- foundation
|   |   |-- foundation.abide.js
|   |   |-- foundation.alerts.js
|   |   |-- foundation.clearing.js
|   |   |-- foundation.cookie.js
|   |   |-- foundation.dropdown.js
|   |   |-- foundation.forms.js
|   |   |-- foundation.interchange.js
|   |   |-- foundation.joyride.js
|   |   |-- foundation.js
|   |   |-- foundation.magellan.js
|   |   |-- foundation.orbit.js
|   |   |-- foundation.placeholder.js
|   |   |-- foundation.reveal.js
|   |   |-- foundation.section.js
|   |   |-- foundation.tooltips.js
|   |   `-- foundation.topbar.js
|   |-- foundation.min.js
|   `-- vendor
|       |-- custom.modernizr.js
|       |-- jquery.js
```

[2] http://foundation.zurb.com/download.php

```
|       `-- zepto.js
`-- robots.txt

5 directories, 26 files
```

The content was organised within three main folders: `css`, `js` and `img`. The *minified* variation of CSS style sheets and JS plugins were provided for use in production. `foundation.min.js` contained the core and the rest of JavaScript plugins, but these ones were also provided in separate files so they could be loaded individually.

The second way was available as a Ruby Gem and it required Compass to compile from Sass/SCSS to CSS:

```
$ sudo gem install zurb-foundation
$ sudo gem install compass
```

To create a new project, the command `compass` was used as follows:

```
$ compass create stadion-with-f4 --require zurb-foundation \
--using foundation
```

The project structure was the following:

```
$ tree stadion-with-f4 --charset=ASCII -L 3
stadion-with-f4
|-- MIT-LICENSE.txt
|-- config.rb
|-- humans.txt
|-- index.html
|-- javascripts
|   |-- foundation
|   |   |-- foundation.alerts.js
|   |   |-- foundation.clearing.js
|   |   |-- foundation.cookie.js
|   |   |-- foundation.dropdown.js
|   |   |-- foundation.forms.js
|   |   |-- foundation.joyride.js
|   |   |-- foundation.js
|   |   |-- foundation.magellan.js
|   |   |-- foundation.orbit.js
|   |   |-- foundation.placeholder.js
|   |   |-- foundation.reveal.js
|   |   |-- foundation.section.js
|   |   |-- foundation.tooltips.js
```

```
|   |   `-- foundation.topbar.js
|   `-- vendor
|       |-- custom.modernizr.js
|       |-- jquery.js
|       `-- zepto.js
|-- robots.txt
|-- sass
|   |-- _settings.scss
|   `-- app.scss
`-- stylesheets
    `-- app.css

5 directories, 25 files
```

The main difference with the basic CSS alternative was the existence of three new files:

- `config.rb` contained some configuration properties.
- `_settings.scss` contained variables and mixins to personalise the project.
- `app.scss` was the main SCSS file to be compiled into the corresponding `app.css`.

During the compilation process the SCSS files were copied from the zurb-foundation gem to the local folder `.sass-cache`:

```
$ dirname `gem which zurb-foundation`
/Library/Ruby/Gems/1.8/gems/zurb-foundation-4.1.6/lib
$ ls $_/../scss/foundation/components
_alert-boxes.scss   _grid.scss      _section.scss
_block-grid.scss    _inline-lists.scss  _side-nav.scss
_breadcrumbs.scss   _joyride.scss       _split-buttons.scss
_button-groups.scss _keystrokes.scss    _sub-nav.scss
_buttons.scss       _labels.scss        _switch.scss
_clearing.scss      _magellan.scss      _tables.scss
_custom-forms.scss  _orbit.scss     _thumbs.scss
_dropdown-buttons.scss  _pagination.scss    _tooltips.scss
_dropdown.scss      _panels.scss        _top-bar.scss
_flex-video.scss    _pricing-tables.scss    _type.scss
_forms.scss     _progress-bars.scss _visibility.scss
_global.scss        _reveal.scss
$ compass clean
remove .sass-cache/
remove stylesheets/app.css
$ compass compile --output-style compressed
create stylesheets/app.css
```

There was an option to point the zurb-foundation gem to the source code in GitHub by creating a Gemfile with the following content:

```
source "https://rubygems.org"
gem "zurb-foundation", :git => "git@github.com:zurb/foundation.git"
gem "compass"
```

Afterwards, `bundler` was used to install the dependencies:

```
$ bundle install
Fetching git@github.com:zurb/foundation.git
$ bundle exec compass create . --require zurb-foundation \
--using foundation
```

The third way of getting Foundation is through the package manager Bower, as follows:

```
$ bower search foundation
Search results:
    foundation git://github.com/zurb/foundation
    foundation-icons git://github.com/zurb/foundation-icons.git
   components-foundation git://github.com/components/foundation.git
$ bower install foundation
$ bower install foundation-icons
$ tree stadion-with-f4-bower --charset=ASCII -L 3
stadion-with-f4-bower
|-- components
|   `-- foundation
|       |-- CONTRIBUTING.md
|       |-- Gemfile
|       |-- Gemfile.lock
|       |-- Gruntfile.js
|       |-- LICENSE
|       |-- README.md
|       |-- Rakefile
|       |-- bower.json
|       |-- composer.json
|       |-- docs
|       |-- foundation.gemspec
|       |-- js
|       |-- lib
|       |-- package.json
|       |-- scss
|       |-- spec
```

```
|       `-- templates
`-- npm-debug.log

8 directories, 12 files
```

4.2.2.2 Twitter Bootstrap

Twitter Bootstrap offered the following alternatives:

- Plain CSS (default or customised).
- Source code using LESS.
- Using Bower.

The file structure for the default CSS option is shown below:

```
$ tree bootstrap --charset=ASCII -L 2
bootstrap
|-- css
|   |-- bootstrap-responsive.css
|   |-- bootstrap-responsive.min.css
|   |-- bootstrap.css
|   `-- bootstrap.min.css
|-- img
|   |-- glyphicons-halflings-white.png
|   `-- glyphicons-halflings.png
`-- js
    |-- bootstrap.js
    `-- bootstrap.min.js

3 directories, 8 files
```

Bootstrap, unlike Foundation, distinguished between responsive and regular style sheet files. Even though vendor specific libraries were not provided, jQuery was required for the JavaScript plug-ins to work. On the other hand it included glyphicons-halflings, a set of small icons under the same license as Bootstrap, Apache License v2.0.

The second option for getting Twitter Bootstrap is to download the source code and use LESS to compile the style sheets. The file structure is shown below:

```
$ tree twitter-bootstrap-d9b502d --charset=ASCII -L 1
twitter-bootstrap-d9b502d
|-- CHANGELOG.md
|-- CONTRIBUTING.md
```

```
|-- LICENSE
|-- Makefile
|-- README.md
|-- component.json
|-- composer.json
|-- docs
|-- img
|-- js
|-- less
`-- package.json

4 directories, 8 files
```

The main folders are:

- `docs` containing the documentation of the framework.
- `img` containing the `glyphicons` icons.
- `js` containing the JavaScript plug-ins.
- `less` containing the style sheets in LESS format.

Additionally, the folder `less` contained three special files to personalise the project:

- `variables.less` with variables such as colour and size.
- `bootstrap.less` allowed to import regular components independently.
- `responsive.less` allowed to import responsive components independently.

All 5 dependencies of the project were managed by `npm` and declared in the corresponding `devDependencies` property of the file `package.json`. Therefore, they were only used during the development phase. Below is a brief description of each dependency:

- `connect` was used to serve the static pages during testing.
- `hogan.js` was used as a template engine to create the documentation from `.mustache` files.
- `jshint` was used to validate the JavaScript libraries.
- `recess` was used to compile the LESS files into CSS.
- `uglify` was used to minify the JavaScript libraries.

The project was built by issuing the command `make` as shown below:

```
$ cd twitter-bootstrap-d9b502d
$ make bootstrap
$ tree bootstrap --charset=ASCII -L 2
bootstrap
|-- css
|   |-- bootstrap-responsive.css
|   |-- bootstrap-responsive.min.css
|   |-- bootstrap.css
|   `-- bootstrap.min.css
|-- img
|   |-- glyphicons-halflings-white.png
|   `-- glyphicons-halflings.png
`-- js
    |-- bootstrap.js
    `-- bootstrap.min.js

3 directories, 8 files
```

The third and last way of getting Twitter Bootstrap was through Bower:

```
$ mkdir stadion-with-bootstrap
$ cd $_
$ bower install bootstrap
$ tree components --charset=ASCII -L 2
components
|-- bootstrap
|   |-- CHANGELOG.md
|   |-- CONTRIBUTING.md
|   |-- LICENSE
|   |-- Makefile
|   |-- README.md
|   |-- component.json
|   |-- composer.json
|   |-- docs
|   |-- img
|   |-- js
|   |-- less
|   `-- package.json
`-- jquery
    |-- README.md
    |-- component.json
    |-- composer.json
    |-- jquery.js
    |-- jquery.min.js
    `-- package.json
```

```
7 directories, 14 files
```

The folder `bootstrap` contained the source code of the distribution as if it had been downloaded, since Bower used the source repository GitHub.

Twitter Bootstrap was also available with Sass instead of LESS in two different versions, one maintained by Thomas McDonald[3] and the other, by John W. Long[4]. Additionally, Yeoman had two generators, generator-bootstrap and yeoman-foundation, to create responsive applications.

Regarding the size of the default `main.css` generated, Bootstrap was 28 KB bigger than Foundation:

```
$ du -h dist/styles/bootstrap-min.css
120K    dist/styles/bootstrap-min.css
$ du -h dist/styles/foundation-min.css
 92K    dist/styles/foundation-min.css
```

In this study, Twitter Bootstrap was chosen over Zurb Foundation because it had a slightly better *look & feel*. However, both frameworks were a good option to build maintainable and responsive web sites through Bower. They both had several options to personalise a website, such as the use of a file with variables and a file to import the different components. The main difference between the two was the language used to generate the style sheets. Zurb Foundation was based on Sass and used the Ruby gem `compass` to compile into CSS. While Twitter Bootstrap was based on LESS and used `make` along with a set of tools to validate, generate the *minified* version of the JavaScript components and compile into CSS. Another difference was that Foundation included responsive support right out of the box, while in Bootstrap it was available through separate files. Twitter provided the following explanation:

> Bootstrap doesn't include responsive features by default at this time as not everything needs to be responsive. Instead of encouraging developers to remove this feature, we figure it best to enable it as needed.

The next section explains the build process of a JavaScript application from development to production.

4.2.3 Build Process

In the Diaulos application, the build lifecycle was managed by Maven. For instance, the default lifecycle had the following build phases:

[3] https://github.com/thomas-mcdonald/bootstrap-sass
[4] https://github.com/jlong/sass-twitter-bootstrap

1. validate
2. compile
3. test
4. package
5. verify
6. install
7. deploy

In the JavaScript World, the de-facto build tool is Grunt, a JavaScript Task Runner. Grunt uses the configuration file `Gruntfile.js` to define build phases similar to those from Maven. To understand the build process, it is important to focus for a moment on the file structure of a JavaScript application. The source folder `app` of Stadion had the following content:

- `components` contained external dependencies maintained by Bower such as AngularJS, Twitter Bootstrap, D3.js, jQuery, and so on. Some of them were served from a Content Delivery Network (CDN), especially in the production environment.
- `scripts` contained JavaScript files or any other programming language that trans-compiles to JavaScript such as CoffeeScript. As it will be explained later, the build process should include the validation of the files with JSHint.
- `styles` contained CSS files or any other programming language that is interpreted into CSS like LESS or Sass. In the production environment, it is recommended to generate a minified version of the style sheets.
- `views` contained HTML files or any other template engine, such as Jade. Even though Express could compile `.jade` files into `.html` at runtime in the server, they were compiled before hand in the build process to gain speed.

The configuration file `Gruntfile.js` defined the following set of tasks for the build phase:

```
grunt.registerTask('build', [
    'clean:dist',
    'jshint',
    'test',
    'coffee',
    'compass:dist',
    'less',
    'jade:dist',
    'useminPrepare',
    'imagemin',
    'cssmin',
```

```
    'htmlmin',
    'concat',
    'copy',
    'cdnify',
    'ngmin',
    'uglify',
    'rev',
    'usemin'
]);
```

Below is an explanation of each task:

1. `clean:dist`: This task deleted the `dist` folder as defined in the grunt-contrib-clean project.
2. `jshint`: grunt-contrib-jshint was used as a wrapper for JSHint, a tool to detect potential error in JavaScript.
3. `test`: Test was actually a set of tasks defined in `Gruntfile.js`.
4. `coffee`: It compiled from CoffeeScript to JavaScript as explained in the grunt-contrib-coffee project.
5. `compass:dist`: grunt-contrib-compass was used to compile Sass files into CSS.
6. `less`: Similar to the previous task, grunt-contrib-less was used to compile LESS into CSS.
7. `jade:dist`: grunt-contrib-jade generated HTML files from the Jade engine template.
8. `useminPrepare`: This task was part of grunt-usemin which detected special comment blocks in HTML and updated `cssmin`, `concat`, `uglify` and `requirejs` tasks accordingly.
9. `imagemin`: It was used to minify images using OptiPNG and jpegtran as defined in the project grunt-contrib-imagemin.
10. `cssmin`: grunt-contrib-cssmin concatenated several CSS files and generated a reduced version.
11. `htmlmin`: grunt-contrib-htmlmin performed several tasks to HTML files in order to generate a reduced version, such as removing comments and white spaces.
12. `concat`: It concatenated files with several options defined in grunt-contrib-concat. For example, it could replace all 'use strict' statements in the code with a single one at the top.
13. `copy`: grunt-contrib-copy was used to copy files and directories.
14. `cdnify`: grunt-google-cdn replaced references to resources on the Google CDN.
15. `ngmin`: grunt-ngmin was a wrapper of ngmin which transformed AngularJS code so it could later be minified.
16. `uglify`: grunt-contrib-uglify was based on UglifyJS, which reduced the size of JavaScript by replacing the name of variables for a shorter version.

17. `rev`: grunt-rev renamed static files with a unique name that depended on their contents. This allowed the TTL of the cache for these files to be infinitely extended.
18. `usemin`: It replaced the comment blocks found with `useminPrepare` by a reference to a single file, as well as all references to images, scripts and CSS files by their minified version.

4.3 Cloud Deployment

Heroku was used in chapter 3 to deploy the solution architecture based on Java. The process to deploy *Stadion*, our NodeJS-based application, was very similar. Heroku Toolbelt, which provides the command-line tool `heroku`, was already installed. It was not necessary to login via `heroku login`, either. Like in chapter 3, a text file `Procfile` was created to define which command should be executed to start a web `dyno`:

```
$ cat Procfile
web: node server.js
```

The next step was to create an application in Heroku:

```
$ heroku create --region eu
Creating vast-mountain-1850... done, region is eu
http://vast-mountain-1850.herokuapp.com/
git@heroku.com:vast-mountain-1850.git
Git remote heroku added
```

The new repository was added to Git, as shown below:

```
$ git remote -v
heroku  git@heroku.com:vast-mountain-1850.git (fetch)
heroku  git@heroku.com:vast-mountain-1850.git (push)
origin  git@github.com:atreceno/stadion.git (fetch)
origin  git@github.com:atreceno/stadion.git (push)
```

For our convenience, the name of the Git repository in Heroku was renamed to the name of the application as follows:

```
$ git remote rm heroku
$ git remote add heroku git@heroku.com:stadion.git
$ git remote -v
heroku  git@heroku.com:stadion.git (fetch)
heroku  git@heroku.com:stadion.git (push)
origin  git@github.com:atreceno/stadion.git (fetch)
origin  git@github.com:atreceno/stadion.git (push)
```

The name of the application was also renamed in Heroku Dashboard (see Figure 4.2).

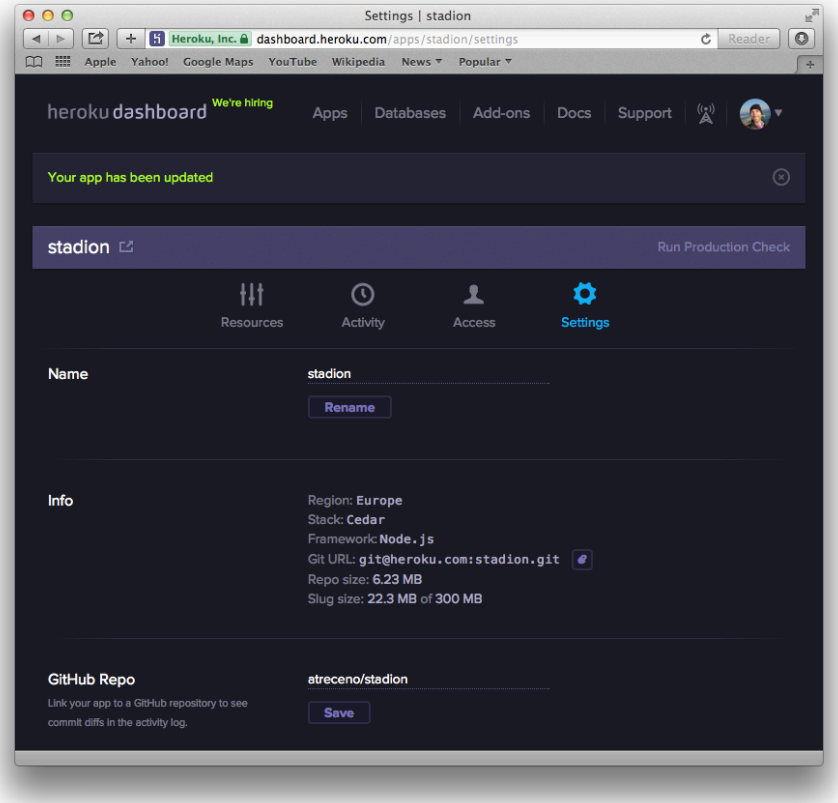

Figure 4.2: Heroku Dashboard - Stadion

A MongoDB-as-a-Service was added from the command line:

```
$ heroku addons:add mongolab
$ heroku config
=== stadion Config Vars
MONGOLAB_URI:
mongodb://surreyquays:london@ds059958.mongolab.com:59958/stadion
NODE_ENV: production
```

Once the application was created through the Dashboard and the file `Procfile` added to Git, *Stadion* could be deployed to Heroku as follows:

```
$ git push heroku master
$ heroku ps:scale web=1
$ heroku ps
=== web (1X): `node server.js`
web.1: idle 2013/11/23 14:06:24 (~ 19h ago)
$ heroku logs --tail
21:17:41 heroku[web.1]: Starting process with `node server.js`
21:17:42 heroku[web.1]: State changed from starting to up
21:17:42 app[web.1]: Express server listening on port 31780
21:17:42 app[web.1]: Loading configuration for production env.
21:17:42 app[web.1]: Mongoose version: 3.6.20
21:17:43 app[web.1]: Connection to DB established
21:18:27 heroku[router]: at=info method=GET path=/
21:18:27 app[web.1]: GET / 200 12ms -
```

Figure 4.3 shows the application up and running.

4.4 Testing

Testing a JavaScript application is not much different from testing a Java application, and many of the concepts introduced in section 3.4 are still valid for this section.

4.4.1 Unit Tests

In this case, Karma, one of the most popular test runners for JavaScript, was used. It was specifically designed to be used with AngularJS. Karma was updated through `npm` as follows:

```
$ npm ll -g karma
/usr/local/lib
karma@0.10.4
Spectacular Test Runner for JavaScript.
git://github.com/karma-runner/karma.git
http://karma-runner.github.io/

$ npm view karma version
0.10.5
$ npm update -g karma
karma@0.10.5 /usr/local/lib/node_modules/karma
```

Before starting any tests, Karma required a configuration file that was generated with the command `karma init`:

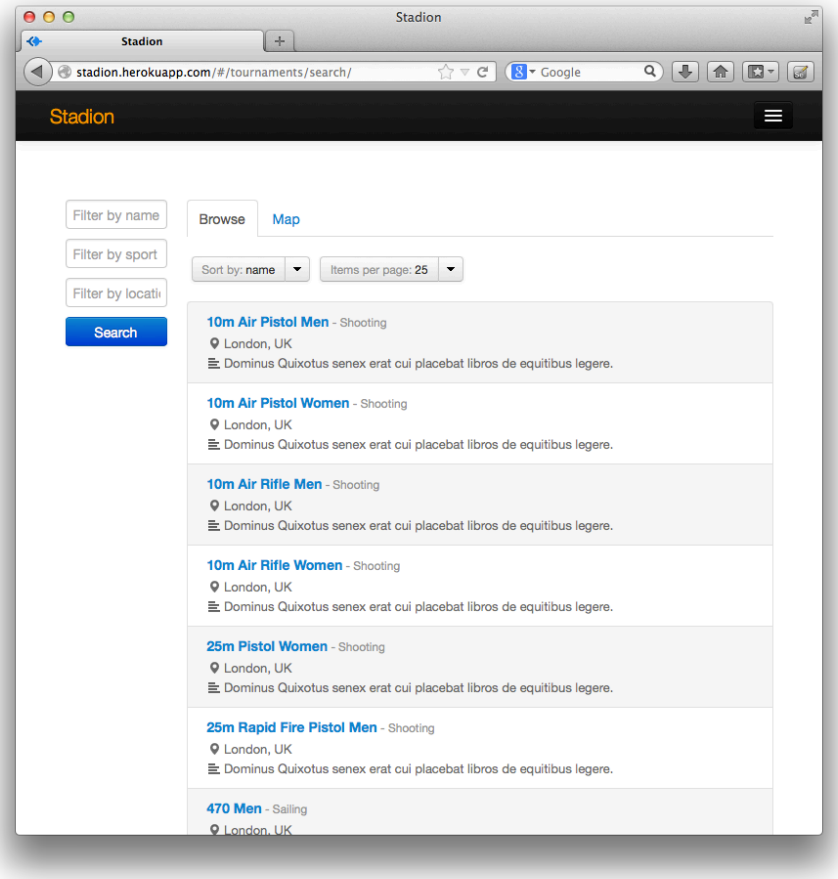

Figure 4.3: Stadion deployed to Heroku

```
$ karma init
Config file generated at "~/dev/git/stadion/karma.conf.js".
```

The content of the file is shown in the following listing:

```
// Generated on Sun Nov 24 2013 12:53:54 GMT+0000 (GMT)
module.exports = function(config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine'],
    files: [
      'app/components/angular/angular.js',
      'app/components/angular-resource/angular-resource.js',
      'app/components/angular-mocks/angular-mocks.js',
      'app/components/angular-bootstrap/ui-bootstrap-tpls.js',
      'app/components/d3/d3.min.js',
      'app/scripts/*.js',
      'app/scripts/**/*.js',
      'test/spec/**/*.js'
    ],
    exclude: [
      '**/*.swp'
    ],
    reporters: ['progress'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome', 'Firefox'],
    captureTimeout: 60000,
    singleRun: false
  });
};
```

The parameter `files` contained the list of files to load in the browser, including the test files and the libraries to run the tests. The paths were relative to the value of `basePath`, which at the same time was relative to the location of the configuration file. The parameter `frameworks` specified the test frameworks to use and it could be any of Jasmine, Mocha and QUnit. In this case, the first one was used to test *Stadion*. Jasmine is a Behavior-Driven Development (BDD) framework for testing JavaScript code in real browsers.

The tests were located in the `test` folder:

```
$ tree test --charset=ASCII
```

```
test
|-- runner.html
`-- spec
    |-- controllers
    |   |-- country.js
    |   |-- main.js
    |   `-- tournament.js
    |-- directives
    |   `-- stMedal.js
    `-- services
        |-- country.js
        `-- socket.js
```

As an example, the test of the `TournamentListCtrl` controller is shown below:

```
describe('TournamentListCtrl controller', function () {
    var scope;
    var sports = [{name:'Archery'},{name:'Athletics'}];
    var tourn = [
        {
            'name' : '10m Air Pistol Men',
            'sport' : 'Shooting',
            'description' : 'Dominus Quixotus',
            'location' : 'London, UK'
        },
        {
            'name' : '10m Air Pistol Women',
            'sport' : 'Shooting',
            'description' : 'Dominus Quixotus',
            'location' : 'London, UK'
        }
    ];
    var mongoBaseUrl = 'https://api.mongolab.com/databases/' +
        'stadion/collections';
    var mongoApiKey = '?apiKey=ay7qWRojBsbhp10CJegJkTJ';

    // Load the module
    beforeEach(angular.mock.module('stadion'));

    // Initialize the controller and inject a new scope
    beforeEach(angular.mock.inject(
    function ($controller, $rootScope, $httpBackend) {
        $httpBackend.when('GET', mongoBaseUrl +
            '/sports' + mongoApiKey).respond(sports);
        $httpBackend.when('GET', mongoBaseUrl +
```

```
            '/tournaments' + mongoApiKey).respond(tourn);
        scope = $rootScope.$new();
        $controller('TournamentListCtrl', {$scope: scope});
        $httpBackend.flush();
    }));

    it('should allow you to order by name, location or sport',
    function() {
        expect(scope.predicates.length).toBe(3);
        expect(scope.predicates[0]).toBe('name');
        expect(scope.predicates).toEqual(['name', 'location',
        'sport']);
    });

    it('should allow you to change the page size', function() {
        expect(scope.itemsPerPage.length).toBe(3);
        expect(scope.itemsPerPage[0]).toBe(25);
        expect(scope.itemsPerPage).toEqual([25, 50, 100]);
    });

    it('should allow you to filter by location', function() {
        expect(scope.locations.length).toBeGreaterThan(1);
        expect(scope.locations).toContain('London, UK');
    });

    it('should attach a list of sports to the scope',
    function() {
        expect(scope.sports.length).toBe(3);
        expect(scope.sports[0].name).toBe('Archery');
        expect(scope.sports[1].name).toBe('Athletics');
        expect(scope.sports[2].name).toBe('Badminton');
    });

    it('should attach a list of tournaments to the scope',
    function() {
        expect(scope.tournaments.length).toBe(2);
        expect(scope.tournaments[0].name).toBe(tourn[0].name);
        expect(scope.tournaments[0].sport).toBe(tourn[0].sport);
        expect(scope.tournaments[1].name).toBe(tourn[1].name);
        expect(scope.tournaments[1].sport).toBe(tourn[1].sport);
    });
});
```

To run the tests the command `karma start` was used as follows:

```
$ karma start
```

```
INFO [karma]: Karma server started at http://localhost:9876/
INFO [launcher]: Starting browser Chrome
INFO [launcher]: Starting browser Firefox
INFO [Chrome 31.0.1650 (Mac OS X 10.7.5)]: Connected
INFO [Firefox 25.0.0 (Mac OS X 10.7)]: Connected
Chrome 31.0: Executed 13 of 13 SUCCESS (0.489 secs / 0.156 secs)
Firefox 25.0: Executed 13 of 13 SUCCESS (0.48 secs / 0.17 secs)
TOTAL: 26 SUCCESS
```

All 26 tests were successful.

4.4.2 End-to-End Tests

End-to-End (E2E) tests could perfectly validate both front-end unit tests and integration tests. Selenium was used again to create the test suite and test cases. As an example, the test case for tournaments is shown in the next listing:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html>
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html />
<link rel="selenium.base" href="http://stadion.herokuapp.com/" />
<title>tournament</title>
</head>
<body>
    <table cellpadding="1" cellspacing="1" border="1">
        <thead>
            <tr>
                <td rowspan="1" colspan="3">tournament</td>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>open</td>
                <td>/</td>
                <td></td>
            </tr>
            <tr>
                <td>click</td>
                <td>link=Tournaments</td>
                <td></td>
            </tr>
            <tr>
```

```
                <td>click</td>
                <td>link=Search tournaments</td>
                <td></td>
            </tr>
            <tr>
                <td>sendKeys</td>
                <td>//input[@data-ng-model='tName']</td>
                <td>pistol</td>
            </tr>
            <tr>
                <td>sendKeys</td>
                <td>//input[@data-ng-model='tSport']</td>
                <td>Shooting</td>
            </tr>
            <tr>
                <td>sendKeys</td>
                <td>//input[@data-ng-model='tLocation']</td>
                <td>London, UK</td>
            </tr>
            <tr>
                <td>click</td>
                <td>link=Search</td>
                <td></td>
            </tr>
            <tr>
                <td>verifyText</td>
                <td>//h5[1]/a</td>
                <td>10m Air Pistol Men</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

Figure 4.4 shows the tests were run successfully.

Even though Selenium is valid to perform some functional tests, it was not designed to test the asynchronous nature of an AngularJS application. Performing the same tests at the fastest speed will make the tests fail because the command `clickAndWait` only works when the whole page is reloaded. A recent framework that tackles this issue is Protractor. Protractor is an E2E test framework for AngularJS applications based on WebDriverJS.

The RESTful API was tested with the tool `curl`. To create a new tournament, a `POST` request was sent to the corresponding URL as follows:

```
$ curl -i -X POST -H "Content-type: application/json" \
```

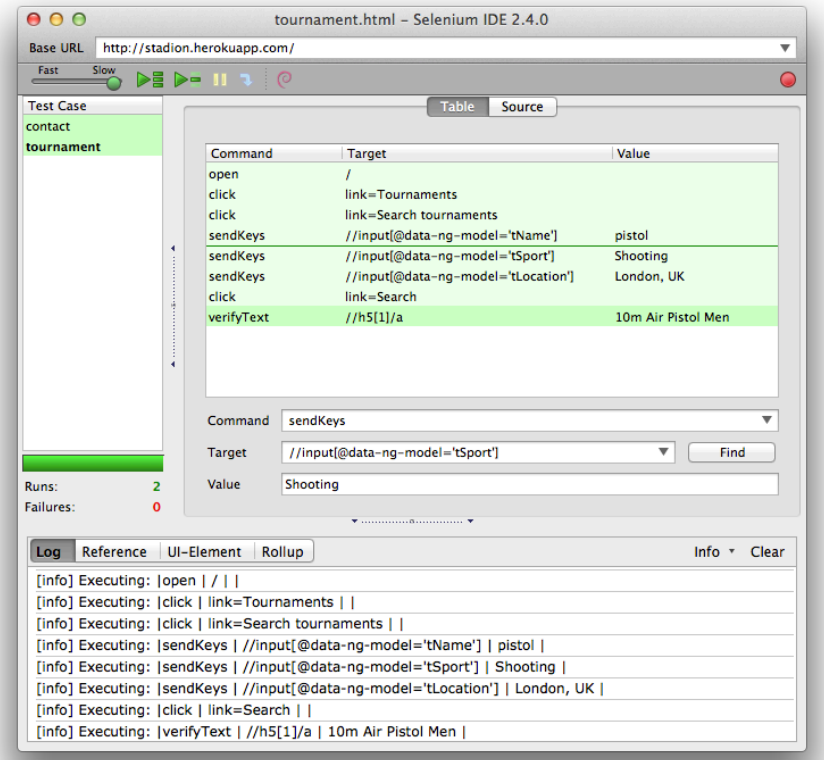

Figure 4.4: E2E Tests with Selenium - Stadion

```
-d @tournament.json http://stadion.herokuapp.com/api/tournaments
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Sat, 30 Nov 2013 14:45:02 GMT
Set-Cookie: connect.sid=Gh6UlFUEqzPJBquvXK5eDg; Path=/; HttpOnly
X-Powered-By: Express
Content-Length: 437
Connection: keep-alive
```

Where `tournament.json` was a file with a tournament in JSON format:

```
{
  "code": "CMW021.Test",
  "name": "Women's Cross-country",
  "sport": "Cycling Mountain Bike",
  "description": "Dominus Quixotus senex erat cui placebat...",
  "location": "London, UK",
  "startDate": "2013-08-01",
  "phases": [
    {
      "code": "CMW0211",
      "name": "Women's Cross-country",
      "fixtures": [
        {
          "code": "CMW021101",
          "name": "Women's Cross-country",
          "date": "2013-07-13T09:54:05.507Z"
        }
      ]
    }
  ],
  "competitors": [],
}
```

The `_id` and `__v` fields were assigned by Mongoose. With the ID, it was possible to retrieve the tournament:

```
$ curl -i \
stadion.herokuapp.com/api/tournaments/5299f9eecf7d280200000003
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Sat, 30 Nov 2013 15:01:47 GMT
Set-Cookie: connect.sid=XOeGZvRL6vhuGxehwJwr1Q; Path=/; HttpOnly
X-Powered-By: Express
Content-Length: 437
```

```
Connection: keep-alive

{
  "_id": {
    "$oid": "5299f9eecf7d280200000003"
  },
  "code": "CMW021.Test",
  "name": "Women's Cross-country",
  "sport": "Cycling Mountain Bike",
  "description": "Dominus Quixotus senex erat cui placebat...",
  "location": "London, UK",
  "startDate": "2013-08-01",
  "phases": [
    {
      "code": "CMW0211",
      "name": "Women's Cross-country",
      "fixtures": [
        {
          "code": "CMW021101",
          "name": "Women's Cross-country",
          "date": "2013-07-13T09:54:05.507Z"
        }
      ]
    }
  ],
  "competitors": [],
  "__v": 0
}
```

To modify the tournament, a `PUT` request was sent, as follows:

```
$ curl -i -X PUT -H "Content-type: application/json" \
-d "{\"name\":\"Women's Mountain Biking CX\"}" \
stadion.herokuapp.com/api/tournaments/5299f9eecf7d280200000003
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Sat, 30 Nov 2013 15:20:27 GMT
Set-Cookie: connect.sid=BmpRWBxlj2U2C2hg%2F5Co; Path=/; HttpOnly
X-Powered-By: Express
Content-Length: 442
Connection: keep-alive

{
  "_id": {
    "$oid": "5299f9eecf7d280200000003"
```

```
  },
  "code": "CMW021.Test",
  "name": "Women's Mountain Biking CX",
  "sport": "Cycling Mountain Bike",
  "description": "Dominus Quixotus senex erat cui placebat...",
  "location": "London, UK",
  "startDate": "2013-08-01",
  "phases": [
    {
      "code": "CMW0211",
      "name": "Women's Cross-country",
      "fixtures": [
        {
          "code": "CMW021101",
          "name": "Women's Cross-country",
          "date": "2013-07-13T09:54:05.507Z"
        }
      ]
    }
  ],
  "competitors": [],
  "__v": 0
}
```

A `DELETE` request is shown below:

```
$ curl -i -X DELETE \
stadion.herokuapp.com/api/tournaments/5299f9eecf7d280200000003
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Sat, 30 Nov 2013 15:25:47 GMT
Set-Cookie: connect.sid=sI7D2SrkKByBM22GLJxoUw; Path=/; HttpOnly
X-Powered-By: Express
Content-Length: 442
Connection: keep-alive
```

4.4.3 Non-functional Tests

4.4.3.1 Load testing with Siege

Performing non-functional tests on a Single-Page Application (SPA) can be quite challenging. In an SPA application, the user hits the server once, downloads the page and from there, the browser sends XHR requests to the same or different servers. For this reason, it would be unrealistic to perform a load test with the following command:

```
$ siege -c40 -d10 -t40M -q \
"http://stadion.herokuapp.com/#/tournaments/search/"
```

Each time `siege` is sending 40 concurrent requests, it is downloading the home page without triggering any query in the database:

```
$ siege -g "http://stadion.herokuapp.com/#/tournaments/search/"
GET / HTTP/1.0
Host: stadion.herokuapp.com
Accept: */*
User-Agent: Mozilla/5.0 (pc-i686-linux-gnu) Siege/3.0.4
Connection: close

HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Content-Type: text/html; charset=UTF-8
Date: Thu, 28 Nov 2013 11:02:16 GMT
Etag: "6567-1385495010000"
Last-Modified: Tue, 26 Nov 2013 19:43:30 GMT
Set-Cookie: connect.sid=s%3AMawertbsj7Se24lxib; Path=/; HttpOnly
X-Powered-By: Express
Content-Length: 6567
Connection: Close

<!DOCTYPE html><html lang="en"><head><meta charset="utf-8">
(HTML content omitted)
</body></html>
```

A more realistic way would be 40 concurrent users hitting the server only once and sending the rest of the requests to the API URL. This was accomplished by setting "http://stadion.herokuapp.com/" as a login URL in `.siegerc` and then issuing the command `siege` with 40 concurrent users, as follows:

```
$ siege -c40 -d10 -t40M -q -i -f test/siege-users.txt
```

Where `test/siege-users.txt` was a file containing the URLs under test:

```
BASE_URL=stadion.herokuapp.com
http://$(BASE_URL)/api/countries/517fd22ee4b09ee355bfff13
http://$(BASE_URL)/api/countries/517fd2b8e4b081c6633d867b
http://$(BASE_URL)/api/countries/517fd34fe4b09ee355c00012
http://$(BASE_URL)/api/countries/517fd3c6e4b09ee355c00083
http://$(BASE_URL)/api/countries/51aa3846e4b0ff5b2a28462d
```

```
http://$(BASE_URL)/api/tournaments/51e28a47434a4d297195c88e
http://$(BASE_URL)/api/tournaments/51e28a47434a4d297195c88f
http://$(BASE_URL)/api/tournaments/51e28a47434a4d297195c890
http://$(BASE_URL)/api/tournaments/51e28a47434a4d297195c891
(90 lines omitted)
http://$(BASE_URL)/api/tournaments/51e28a38434a4d297195c775
```

Recalling from chapter 3, the number of 40 concurrent users was not taken at random. It was an estimation of the number of customers interested in receiving the Olympic feed. To simulate the competition, the same worst case scenario explained in section 3.4 was used: 30 competitions running at the same time, where each competition was generating 139 results every 15 minutes. This was equivalent to sending 139 results with an average delay of 30 seconds:

```
$ siege -c1 -d60 -t40M -q -f test/siege-run-competition.txt
```

Where the file `test/siege-run-competition.txt` had at least 80 tournaments:

```
BASE_URL=stadion.herokuapp.com/api/tournaments
http://$(BASE_URL)/529a582d58c1080200000001 PUT < results.json
http://$(BASE_URL)/51e28a36434a4d297195c744 PUT < results.json
http://$(BASE_URL)/51e28a36434a4d297195c745 PUT < results.json
http://$(BASE_URL)/51e28a36434a4d297195c746 PUT < results.json
http://$(BASE_URL)/51e28a36434a4d297195c747 PUT < results.json
http://$(BASE_URL)/51e28a36434a4d297195c748 PUT < results.json
http://$(BASE_URL)/51e28a36434a4d297195c749 PUT < results.json
http://$(BASE_URL)/51e28a36434a4d297195c74a PUT < results.json
http://$(BASE_URL)/51e28a36434a4d297195c74b PUT < results.json
(70 lines omitted)
http://$(BASE_URL)/51e28a3b434a4d297195c7a7 PUT < results.json
```

The file `results.json` contained 139 results of a tournament:

```
{"competitors": [
{"seed": 1   , "result": 101, "name": "Competitor 1"},
{"seed": 2   , "result": 102, "name": "Competitor 2"},
{"seed": 3   , "result": 103, "name": "Competitor 3"},
{"seed": 4   , "result": 104, "name": "Competitor 4"},
{"seed": 5   , "result": 105, "name": "Competitor 5"},
{"seed": 6   , "result": 106, "name": "Competitor 6"},
{"seed": 7   , "result": 107, "name": "Competitor 7"},
{"seed": 8   , "result": 108, "name": "Competitor 8"},
(130 lines omitted)
{"seed": 139 , "result": 239, "name": "Competitor 139"}
]}
```

To monitor the application, the New Relic NodeJS agent was used. It was installed via `npm` package manager as follows:

```
$ npm install newrelic --save
newrelic@1.1.1 node_modules/newrelic
```

This agent used the configuration file `newrelic.js` that could be copied from the `node_modules/newrelic` folder:

```
$ cp node_modules/newrelic/newrelic.js .
```

After updating the value of `app_name` and `license_key`, the file's content is shown in the next listing:

```
/**
 * New Relic agent configuration.
 *
 * See lib/config.defaults.js in the agent distribution for a
 * more complete description of configuration variables and their
 * potential values.
 */
exports.config = {
  /**
   * Array of application names.
   */
  app_name : ['Stadion'],

  /**
   * Your New Relic license key.
   */
  license_key : 'aa305c2ede5e34rt45y6530660a06573e2a1',
  logging : {
    /**
     * Level at which to log. 'trace' is most useful to New Relic
     * when diagnosing issues with the agent, 'info' and higher
     * will impose the least overhead on production applications.
     */
    level : 'trace'
  }
};
```

In order for the application *Stadion* to start sending data to New Relic, the line `require('newrelic');` had to be added to the application's startup script `server.js`.

The tests were scheduled in the server with `cron` as follows:

```
$ crontab -l
35 */2 * * * /home/apt/agustin/usr/bin/siege -c40 -d10 \
-t45M -q -i -f test/siege-users.txt \
>> test/siege-report-users.txt 2>&1
35 * * * * /home/apt/agustin/usr/bin/siege -c1 -d60 -t45M -q \
-f test/siege-run-competition.txt \
>> test/siege-report-comp.txt 2>&1
```

A 3 hour window of the `siege-report-users.txt` file is summarised in Table 4.1. All 3 runs showed similar results and the fact that the second run was simulating several competitions did not seem to affect the performance of the application. The duration of each run was 45 minutes with a transaction rate close to 480 RPM (Requests Per Minute). The data transferred varied from run to run because of the Internet mode. Furthermore, it was 10 times bigger than the tests performed in the *Diaulos* application, not only because this time, the tests were 5 minutes longer but because each document in MongoDB represented a whole tournament.

Siege Statistics	Run 1	Run 2	Run 3
Transactions	21107	21090	20953
Availability (%)	100.00	100.00	100.00
Elapsed time (sec)	2699.80	2699.63	2699.70
Data transferred (MB)	91.29	93.38	91.66
Response time (sec)	0.13	0.13	0.13
Transaction rate (1/sec)	7.82	7.81	7.76
Throughput (MB/sec)	0.03	0.03	0.03
Concurrency	0.98	0.99	0.98
Successful transactions	21107	21090	20953
Failed transactions	0	0	0
Longest transaction (sec)	0.41	0.54	0.90
Shortest transaction (sec)	0.08	0.08	0.08

Table 4.1: User simulation - Load Testing - Stadion

The `siege-report-comp.txt` file contained the siege reports for the simulation of the competition. Its content is summarised in Table 4.2. The duration of the test was 45 minutes with a transaction rate close to 2 RPM. Like in the previous

report, the main difference with *Diaulos* is the speed: 0.22 versus 3.92 seconds for the response time and 0.31 versus 5.56 seconds for the longest transaction. The data transferred was 0.86 MB, considerably bigger than the 9 KB sent during the *Diaulos* load test. This is because *Stadion* was sending the modified document back to the client as an acknowledgement. Therefore, the 0.86 MB includes the HTTP headers and data. Unlike *Stadion* , *Diaulos* was sending just the HTTP headers, without any other data.

Siege Statistics	Run 1	Run 2	Run 3
Transactions	N/A	87	N/A
Availability (%)	N/A	100.00	N/A
Elapsed time (sec)	N/A	2699.80	N/A
Data transferred (MB)	N/A	0.86	N/A
Response time (sec)	N/A	0.22	N/A
Transaction rate (1/sec)	N/A	0.03	N/A
Throughput (MB/sec)	N/A	0.00	N/A
Concurrency	N/A	0.01	N/A
Successful transactions	N/A	87	N/A
Failed transactions	N/A	0	N/A
Longest transaction (sec)	N/A	0.31	N/A
Shortest transaction (sec)	N/A	0.15	N/A

Table 4.2: Competition simulation - Load Testing - Stadion

Figure 4.5 shows the average server response time during the tests. It was around 10 ms. during all runs, which is lower than the 18 ms. of the *Diaulos* application.

The throughput, shown in Figure 4.6, had an average value close to 1,000 RPM, double the expected value of 480 RPM (482 RPM if the competition is running). This is because of a bug in New Relic NodeJS agent which doubles the value of requests per minute.

The overall response of the application is excellent with an Apdex score of 1, as shown in Figure 4.7.

Table 4.3 shows the number of hits, the average server response time and the total time spent on each web transaction. The number of transactions were double the expected due to the same bug in New Relic NodeJS agent.

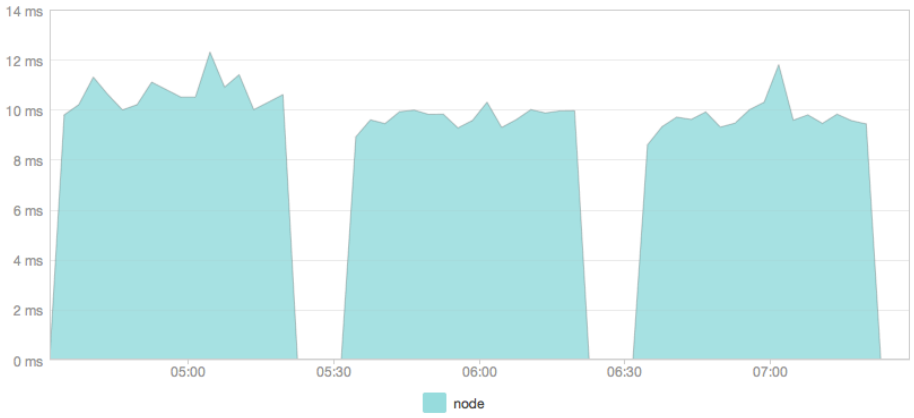

Figure 4.5: Server Response Time - Load Testing - Stadion

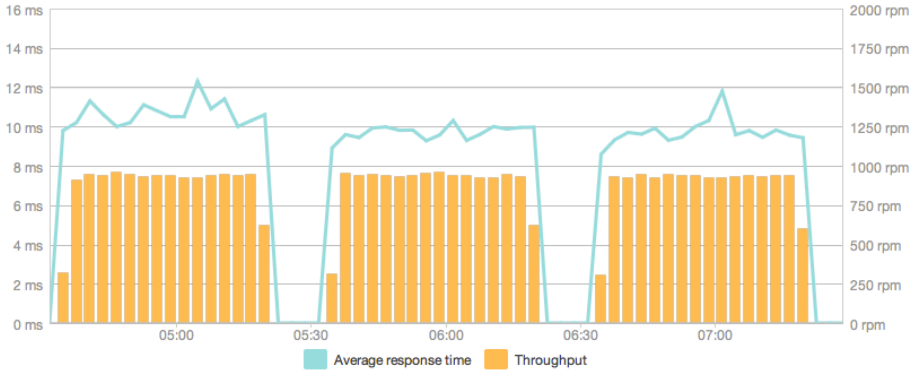

Figure 4.6: Server Throughput - Load Testing - Stadion

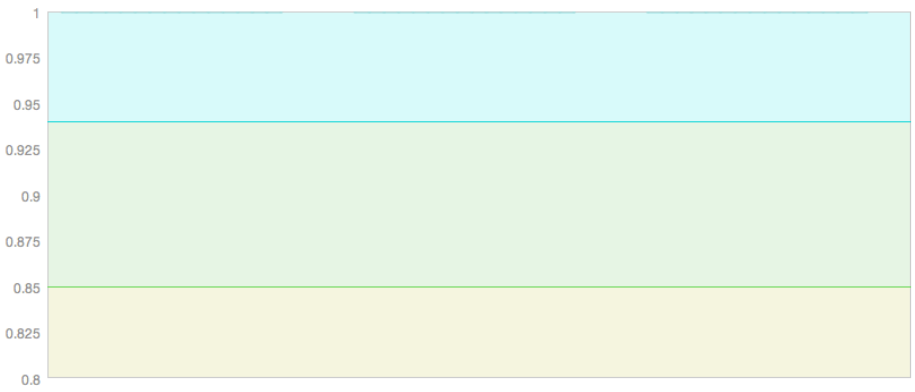

Figure 4.7: Apdex Score - Load Testing - Stadion

Transaction	Count	Avg(ms)	Total(ms)	Total(% time)
get /api/tournaments/:id	63,584	16.1	1,020,000	80.2
get /api/countries/:id	62,480	3.82	239,000	18.8
put /api/tournaments/:id	172	75.9	13,100	1.0
get /	242	2.71	655	0.1
Total	126,478	10.1	1,270,000	100.0

Table 4.3: Web transactions - Load Testing - Stadion

The most time consuming transactions (see Figure 4.8) were `GET` requests of tournaments followed by `GET` requests of countries and `PUT` requests of tournaments. The latter corresponded to the results of the competitions being sent. The transaction of the home page could not be distinguished but there were 40 requests of `/*` at the beginning of each run.

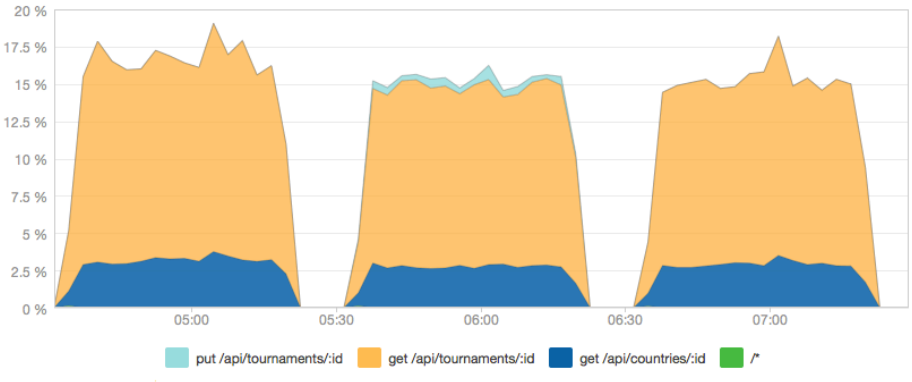

Figure 4.8: Top Web Transactions - Load Testing - Stadion

The database response time can be seen in Figure 4.9. The database throughput had a value of 480 calls per minute (one call per request), as shown in Figure 4.10. Figure 4.11 shows the average time spent on each database operation. The memory usage of the application was stable between 80 MB and 90 MB, much lower than the 350 MB of the *Diaulos* application. Unfortunately, the New Relic NodeJS agent did not allow the CPU usage to be monitored at the moment of writing.

Figure 4.9: Database Response Time - Load Testing - Stadion

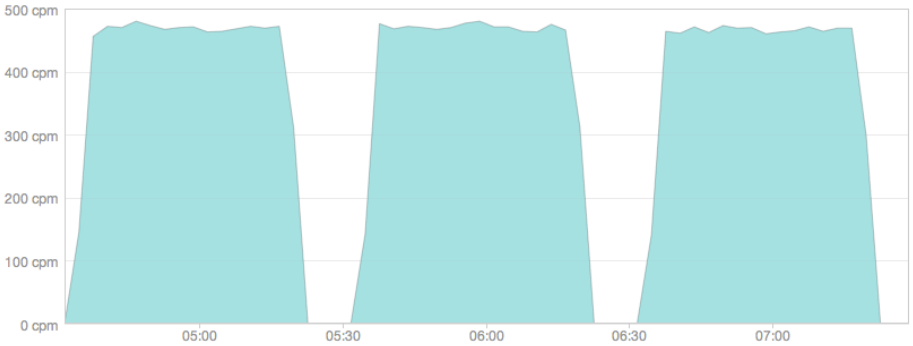

Figure 4.10: Database Throughput - Load Testing - Stadion

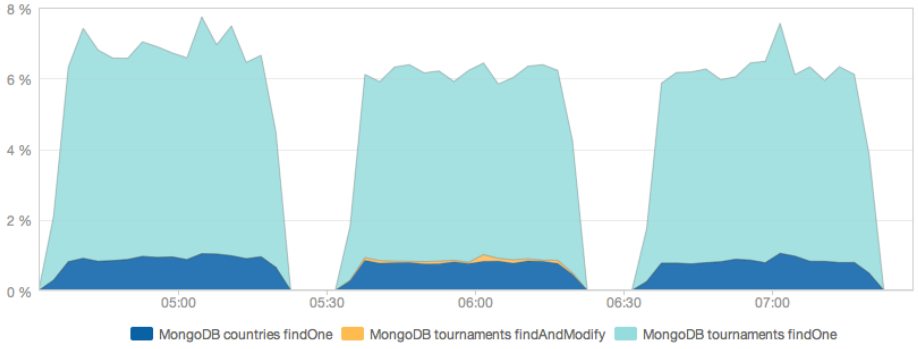

Figure 4.11: Top Database Operations - Load Testing - Stadion

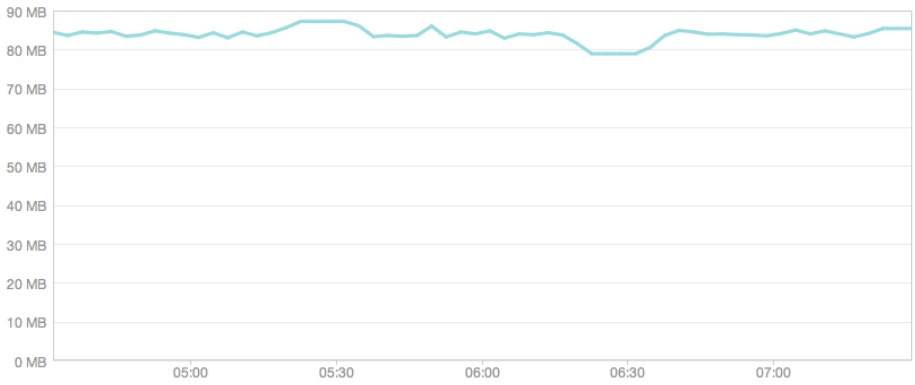

Figure 4.12: Memory Usage - Load Testing - Stadion

4.4.3.2 Stress Testing with Siege

As in the previous chapter, three runs were scheduled using the `cron` utility to simulate 40, 60 and 80 concurrent customers using the REST interface while the competition was running:

```
$ crontab -l
0 1 * * * /home/apt/agustin/usr/bin/siege -c40 -b -t45M -q -i \
-f test/siege-users.txt >> test/siege3.txt 2>&1
0 2 * * * /home/apt/agustin/usr/bin/siege -c60 -b -t45M -q -i \
-f test/siege-users.txt >> test/siege3.txt 2>&1
0 3 * * * /home/apt/agustin/usr/bin/siege -c80 -b -t45M -q -i \
-f test/siege-users.txt >> test/siege3.txt 2>&1
0 1,2,3 * * * /home/apt/agustin/usr/bin/siege -c1 -d60 -t45M -q \
-f test/siege-run-competition.txt >> test/siege4.txt 2>&1
```

The summary report generated by `siege` for the simulation of users is shown in Table 4.4. As with the other tests, all transactions were successful. The transaction rate was slower compared to the *Diaulos* application but the throughput measured in MB/s was higher. The first run had the longest transaction because in this case, the application was in an idle state when the test started.

Table 4.5 shows the summary report corresponding to the simulation of the competitions. The load was the same throughout the tests: 30 simultaneous competitions sending 139 results every 15 minutes. As mentioned in the previous section, the data transferred corresponded to the HTTP header plus the modified tournament included in the response. The more concurrent users there were, the longer the response time. The maximum longest transaction was 1.13 seconds, compared to 24.33 seconds in the *Diaulos* application. The higher number of requests in the *Diaulos* tests were creating a bottleneck in the database, which had only 34 connections available.

Siege Statistics	Run 1	Run 2	Run 3
Transactions	347152	338124	313524
Availability (%)	100.00	100.00	100.00
Elapsed time (sec)	2699.85	2700.00	2699.92
Data transferred (MB)	1520.91	1475.80	1367.24
Response time (sec)	0.31	0.48	0.69
Transaction rate (1/sec)	128.58	125.23	116.12
Throughput (MB/sec)	0.56	0.55	0.51
Concurrency	39.90	59.88	79.87
Successful transactions	347152	338124	313524
Failed transactions	0	0	0
Longest transaction (sec)	12.84	3.62	1.96
Shortest transaction (sec)	0.09	0.10	0.10

Table 4.4: User Simulation - Stress Testing - Stadion

Siege Statistics	Run 1	Run 2	Run 3
Transactions	83	103	74
Availability (%)	100.00	100.00	100.00
Elapsed time (sec)	2699.27	2699.99	2699.99
Data transferred (MB)	0.82	0.99	0.74
Response time (sec)	0.39	0.54	0.74
Transaction rate (1/sec)	0.03	0.04	0.03
Throughput (MB/sec)	0.00	0.00	0.00
Concurrency	0.01	0.02	0.02
Successful transactions	83	103	74
Failed transactions	0	0	0
Longest transaction (sec)	0.81	0.79	1.13
Shortest transaction (sec)	0.26	0.29	0.48

Table 4.5: Competition Simulation - Stress Testing - Stadion

The server response time is shown in Figure 4.13. During load testing there was an average server response time of 10 ms. with the competition running. In benchmarking mode, the response time was 60 ms., 100 ms. and 120 ms. for 40, 60 and 80 concurrent users.

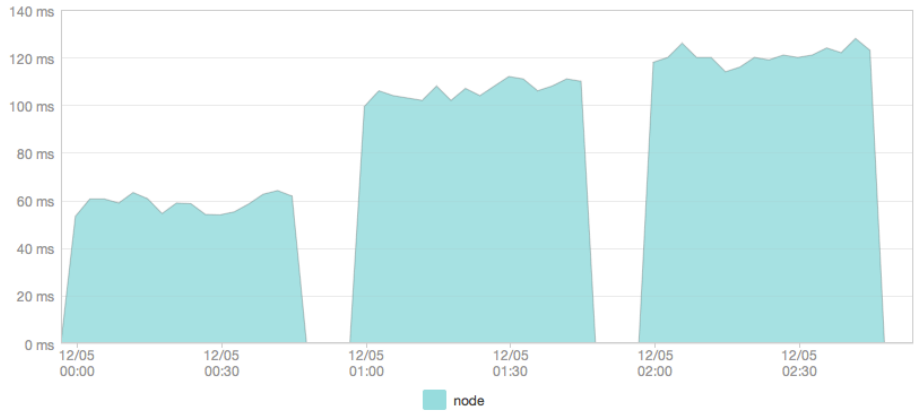

Figure 4.13: Server Response Time - Stress Testing - Stadion

The server throughput is shown in Figure 4.14 with double the number of requests as a result of a bug. It should be around 7500 requests per minute, i.e., 125 requests per second.

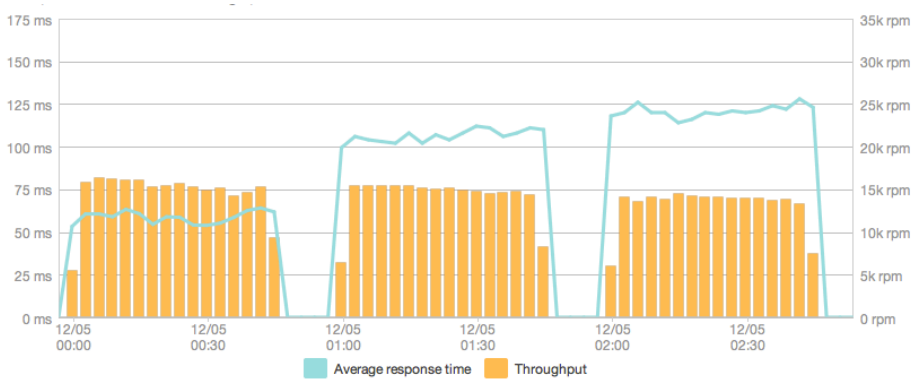

Figure 4.14: Server Throughput - Stress Testing - Stadion

Figure 4.15 shows that the Apdex score is affected by the number of concurrent users, reaching a value close to 0.7 when there are 80 concurrent users. It is still an acceptable value as the Apdex T-value was defined quite low (0.1 seconds).

The web transactions by wall-clock time are shown in Figure 4.16. As expected, the most time consuming web transactions are `GET` requests of tournaments and

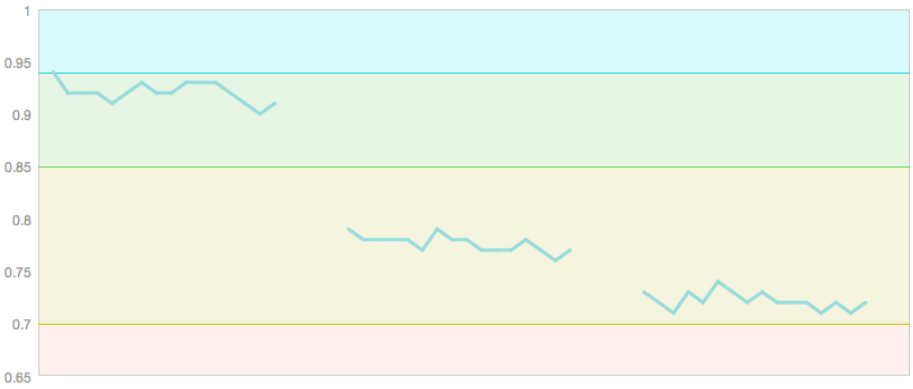

Figure 4.15: Apdex Score - Stress Testing - Stadion

countries because they are queried more often. Even though they are queried the same number of times, tournaments have a higher value because the MongoDB documents are also bigger in size. The other two web transactions, a `PUT` request of tournaments and a `GET` request of the root context, are represented in Figure 4.17. The spike of the `GET` request is because each user hits the home page once and then uses the single-page application. The `POST` requests are distributed evenly because it represents 30 competitions sending 139 results every 15 minutes.

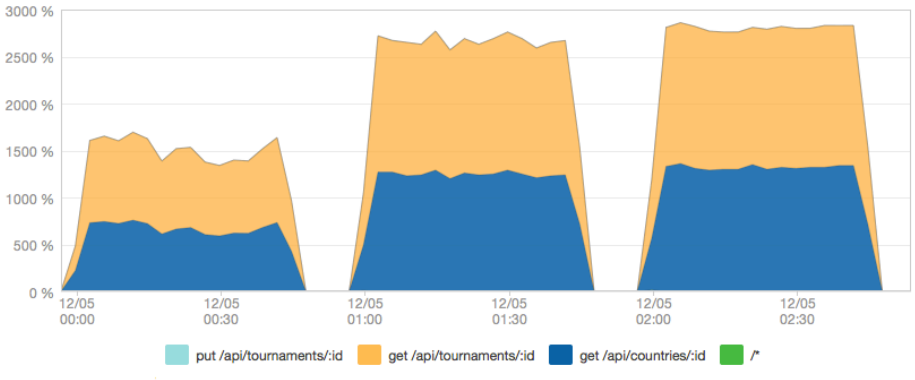

Figure 4.16: Top Web Transactions - Stress Testing - Stadion

The average database response time is shown in Figure 4.18. Comparing figures 4.14 and 4.18, it can be inferred that most of the server response time is spent in the database.

The database throughput was about 7,500 calls per minute (see Figure 4.19). This is an expected value as there is one database call per web transaction.

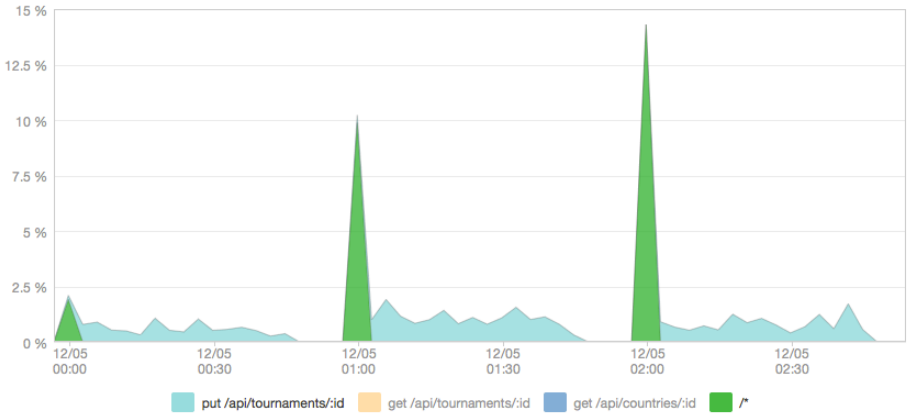

Figure 4.17: Web Transactions (detail) - Stress Testing - Stadion

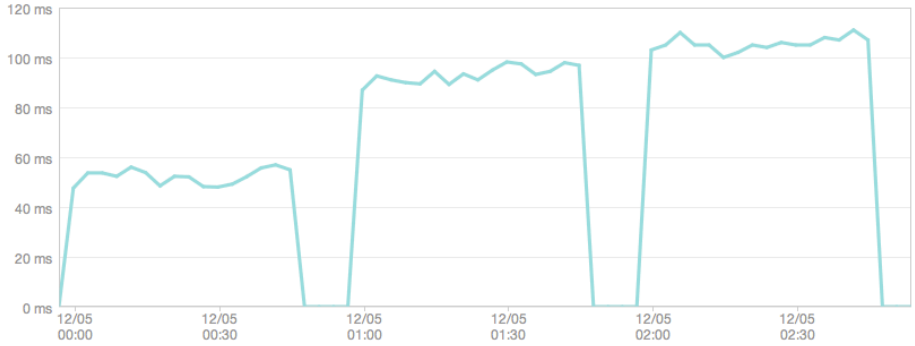

Figure 4.18: Database Response Time - Stress Testing - Stadion

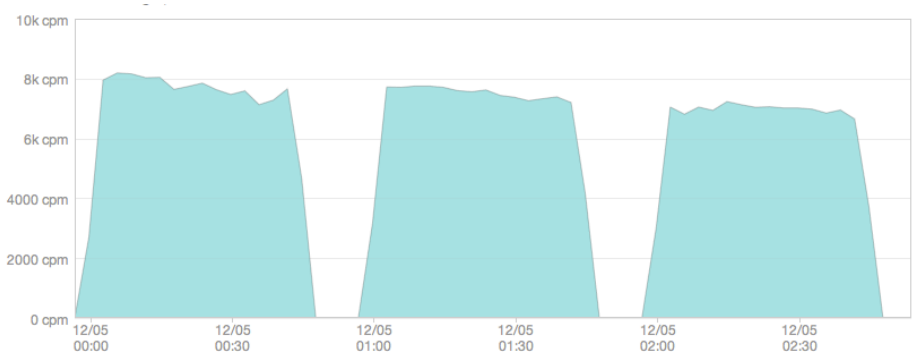

Figure 4.19: Database Throughput - Stress Testing - Stadion

The most time consuming database operations by wall-clock time are shown in Figure 4.20. Again, `findOne()` operation in tournaments and countries collections are the most time consuming because they are more frequent.

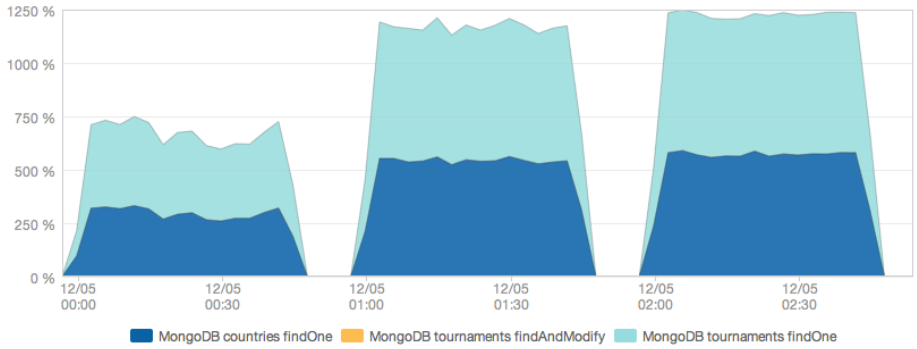

Figure 4.20: Top Database Operations - Stress Testing - Stadion

The memory usage is shown in Figure 4.21. The spike at the beginning of the test was due to the application being in an idle state when the first run started. *Stadion* uses much less memory that *Diaulos*, which was between 400 MB and 500 MB.

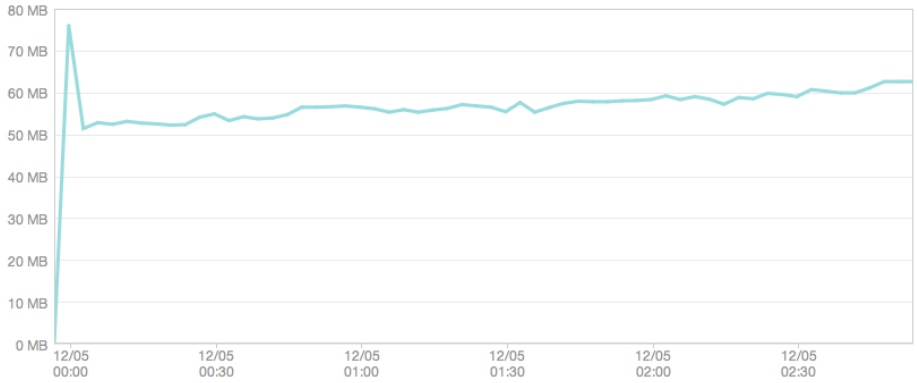

Figure 4.21: Memory Usage - Stress Testing - Stadion

A web *dyno* of 1024 MB had similar results. However, two web *dynos* of 512 MB had faster results but with a few failed transactions as shown in Table 4.6 and Table 4.7.

Siege Statistics	Run 1	Run 2	Run 3
Transactions	542803	553646	566596

Availability (%)	100.00	100.00	100.00
Elapsed time (sec)	2700.07	2699.69	2699.43
Data transferred (MB)	2378.92	2423.17	2474.68
Response time (sec)	0.20	0.29	0.38
Transaction rate (1/sec)	201.03	205.08	209.89
Throughput (MB/sec)	0.88	0.90	0.92
Concurrency	39.79	59.67	79.53
Successful transactions	542803	553646	566596
Failed transactions	4	2	3
Longest transaction (sec)	5.50	5.42	94.20
Shortest transaction (sec)	0.08	0.08	0.08

Table 4.6: User Simulation - Stress Testing (2 web *dynos*)

Siege Statistics	Run 1	Run 2	Run 3
Transactions	82	95	79
Availability (%)	100.00	100.00	100.00
Elapsed time (sec)	2700.01	2700.28	2699.36
Data transferred (MB)	0.81	0.92	0.78
Response time (sec)	0.27	0.33	0.43
Transaction rate (1/sec)	0.03	0.04	0.03
Throughput (MB/sec)	0.00	0.00	0.00
Concurrency	0.01	0.01	0.01
Successful transactions	82	95	79
Failed transactions	0	0	0
Longest transaction (sec)	0.72	0.83	1.57
Shortest transaction (sec)	0.19	0.20	0.16

Table 4.7: Competition Simulation - Stress Testing (2 web *dynos*)

5 Conclusions and Future Work

The whole purpose of education is
to turn mirrors into windows.
—Sydney J. Harris

This chapter includes the conclusions of this study and some recommendations for future work.

5.1 Conclusions

The work described in this study has been concerned with the development of two web-based applications to distribute the results of an Olympic Games to stakeholders. The current approach sends messages that do not make sense on their own unless a previous message has been processed, making the whole system stateful. Customers struggle in designing complicated systems to store and process each message in the correct order. The two solutions presented here allow customers to use a REST interface to interact with the information of the competition. This not only makes the system stateless, but also simplifies the design of the applications for all parties involved.

The first solution architecture is based on Java and uses Spring Roo, a Rapid Application Development (RAD) tool, to create the scaffolding of the application with an important paradigm in mind, convention over configuration. Spring provides the Inversion of Control (IoC) container, and therefore, the application benefits from another good practice, loosely coupled components. Spring MVC and Spring AOP allow the application to separate core and cross-cutting concerns, respectively. On the data persistence layer, two popular choices are backing the application: Hibernate as a JPA provider and MySQL as a database.

The second solution architecture is a Single-Page Application (SPA) that uses JavaScript in both the front-end and the back-end. In the front-end, AngularJS is used to create a Model-View-Controller (MVC) pattern and make the corresponding XMLHttpRequests to a MongoDB service. D3.js is used to generate the medal count chart, and Twitter Bootstrap is also used in the front-end to make the application responsive. This means the application can be used on smartphones right out of the box. In the back-end, NodeJS is used with some frameworks such as Express, Jade and Mongoose. This solution differs from the previous one in the sense that it is more flexible, and manual. Express provides a thin layer of features common to any web application. The template engine Jade allows much more control over HTML code than JSP. This allowed me to design the application according to the standards defined by the World Wide Web Consortium (W3C). Clark et al. (2012) talks about the importance of using web standards. Unfortunately, there are very few companies in the World following this good practice.

It is also worth mentioning the importance of setting the right development environment. Git was used not only to keep the source code under control, but also to write this document. Eclipse, WebStorm and Vi were used as Integrated Development Environments (IDE). MySQL Server, MySQL Workbench, MongoDB, VMWare vFabric tc Server, Jetty, Firefox Aurora, Chrome and Siege among other tools, were installed locally. Regarding the production environment, Cloud Computing offer both, scalability and low cost solutions to deploy applications.

To conclude, new emerging frameworks have changed the landscape of the IT industry in the last few years. However, the design and implementation of Rich Internet Applications remains a difficult task. Even more so when the application has an ambitious purpose such as delivering the results of an Olympic Games. The figures presented in section 2.1 confirm this grade of complexity.

Finally, the following question was formulated in chapter 1: Would it be possible to design a simpler solution by using new emerging frameworks and staying within budget? Throughout the document, it has been demonstrated that the answer to this question is: *yes, it would*.

5.2 Future Work

Even though the two solutions presented here could be used to manage a sporting event the size of an Olympic Games, they could be further developed in a number of ways.

5.2.1 Real-time Results

During a competition, real-time results could be sent to users using push technology such as long polling, Server-Sent Events (SSE) or WebSocket. Wang, Salim, and Moskovits (2012) explains how to use HTML5 and WebSocket. The Atmosphere framework also uses WebSocket and allows to fallback to long polling if the browser does not support it. Atmosphere has several examples of Spring applications, however the Java API for WebSocket has been recently finalised in JSR-356, and it is now part of Java EE 7 (see Lui 2011, chap. 18). Spring Framework 4.0 RC2 (available since December 3, 2013) offers support for WebSocket. For the JavaScript counterpart, socket.io is being replaced by SockJS. Meteor, an open-source platform to build JavaScript applications, uses SockJS to build real-time applications.

5.2.2 Offline Capabilities

The scenario is the following: The web application is being used to manually enter the results of a match, but during the competition the connexion to the server is lost. The application should be able to keep the new results and send them when the connection is back up. This could be implemented through Web Storage providing key-value mapping within the browser. Even though the maximum capacity is different for each browser, the specification recommends to limit the capacity to 5 MB. The JSON file used in chapter 4 to send 139 results was about 10 KB, so 5 MB of capacity should be enough to run 30 competitions simultaneously for more than 4 hours. Other client-side storage techniques are Web SQL Database, Indexed Database and File Access.

5.2.3 Speeding-up Results

Once a competition is finished, it is very unlikely that the results are going to be changed. In this case, a reverse proxy could be used to improve the server response time by caching static and dynamic content. This is also known as web accelerator. Squid is a popular proxy server licensed under the GNU GPL that could be used.

Working with Git

Learning Git can sometimes become a challenge with so many commands and options. The following listing shows the usage of the command `git` and the most commonly used commands:

```
$ git
usage: git [--version] [--exec-path[=<path>]] [--html-path]
           [--man-path] [--info-path] [-p|--paginate|--no-pager]
           [--no-replace-objects] [--bare] [--git-dir=<path>]
           [--work-tree=<path>] [--namespace=<name>]
           [-c name=value] [--help] <command> [<args>]

The most commonly used git commands are:
   add        Add file contents to the index.
   bisect     Find by binary search the change that introduced a
              bug.
   branch     List, create, or delete branches.
   checkout   Checkout a branch or paths to the working tree.
   clone      Clone a repository into a new directory.
   commit     Record changes to the repository.
   diff       Show changes between commits, commit and working
              tree, etc.
   fetch      Download objects and refs from another repository.
   grep       Print lines matching a pattern.
   init       Create an empty git repository or reinitialize an
              existing one.
   log        Show commit logs.
   merge      Join two or more development histories together.
   mv         Move or rename a file, a directory, or a symlink.
```

```
   pull        Fetch from and merge with another repository or a
               local branch.
   push        Update remote refs along with associated objects.
   rebase      Forward-port local commits to the updated upstream
               head.
   reset       Reset current HEAD to the specified state.
   rm          Remove files from the working tree and from the
               index.
   show        Show various types of objects.
   status      Show the working tree status.
   tag         Create, list, delete or verify a tag object signed
               with GPG.

See 'git help <command>' for more information on a specific
command.
```

The purpose of this annex is to breakdown one of the most common workflows used in version control systems.

A.1 The Basics

Git and Mercurial are both examples of distributed Version Control Systems (VCS) as opposed to centralised VCS such as CVS or Subversion. One of the main advantages of the former over the latter is the lack of a single point of failure. In a centralised VCS, when the server fails, nobody can commit changes until the system is up again. If the system cannot be recovered, all the previous history of the project is lost and the only thing available is a local copy of the current working directory. On the downside, a distributed VCS takes longer to clone a repository because it has to fetch all the revision history of the files. Furthermore, it lacks a locking mechanism like some centralised systems have.

Another difference between Git and other revision control systems such as CVS or Subversion is how data is stored. While CVS and Subversion store a base version of each file and incremental changes, Git stores data as snapshots using links if the file has not changed. This will play an important role when using branches.

Git also comes with built-in GUI tools for committing (`git-gui`) and browsing (`gitk`) but there are other third-party tools available. Once installed, Git can be customised by using the command `git config`. By default, it will write the configuration locally to the repository folder, but the options `--global` and `--system` can be used to save the configuration in the home folder of the user or in the installation folder, respectively. A good place to find more about the configuration and Git in general is the GitHub cheat-sheet available online[1].

[1] https://help.github.com/articles/git-cheatsheet

The following listing shows the final configuration used:

```
$ cat ~/.gitconfig
[user]
    name = Agustin Treceno
    email = agustin.treceno@gmail.com
[color]
    ui = auto
[color "branch"]
    current = yellow reverse
    local = yellow
    remote = green
[color "diff"]
    meta = yellow bold
    frag = magenta bold
    old = red bold
    new = green bold
    whitespace = red reverse
[color "status"]
    added = yellow
    changed = green
    untracked = cyan
[core]
    whitespace=fix,-indent-with-non-tab,trailing-space,cr-at-eol
[alias]
    st = status
    ci = commit
    br = branch
    co = checkout
    df = diff
    dc = diff --cached
    lg = log -p
    lol = log --graph --decorate --pretty=oneline --abbrev-commit
    lola = lol --all
    ls = ls-files
    ign = ls-files -o -i --exclude-standard
```

A.2 Initialising a Repository

In order to demonstrate how to use `git`, a simple Java application will be implemented. Specifically, this will be an implementation of the linear search algorithm. Using Maven archetype:generate goal, it is very easy to create the scaffolding of the project:

```
$ cd ~/Development/git
$ mvn archetype:generate \
-DgroupId=com.atreceno.it.tools \
-DartifactId=dummy \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DinteractiveMode=false
$ tree dummy --charset=ASCII
dummy
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- atreceno
    |               `-- it
    |                   `-- tools
    |                       `-- App.java
    `-- test
        `-- java
            `-- com
                `-- atreceno
                    `-- it
                        `-- tools
                            `-- AppTest.java

13 directories, 3 files
```

There is also an interactive mode:

```
$ mvn archetype:generate -Dfilter=org.apache:quickstart
```

In this case, the parameter `filter` is filtering by groupId `org.apache` and artifactId `quickstart`. For more information about Maven, there is a good book available online[2]. Once the project scaffolding has been defined, the command `git init` will create the folder `.git` with an empty Git repository, but it will not add any files to the repository until `git add` is used:

```
$ cd ~/Development/git/dummy
$ git init
$ du -h .git
  0B    .git/branches
 36K    .git/hooks
4.0K    .git/info
```

[2] http://books.sonatype.com/mvnref-book/reference/

```
  0B    .git/objects/info
  0B    .git/objects/pack
  0B    .git/objects
  0B    .git/refs/heads
  0B    .git/refs/tags
  0B    .git/refs
 52K    .git
$ git add .
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   pom.xml
#   new file:   src/main/java/com/atreceno/it/tools/App.java
#   new file:   src/test/java/com/atreceno/it/tools/AppTest.java
#
$ git commit -m "Add scaffolding created by maven"
[master (root-commit) c04c492] Add scaffolding created by maven
 3 files changed, 69 insertions(+)
 create mode 100644 pom.xml
 create mode 100644 src/main/java/com/.../it/tools/App.java
 create mode 100644 src/test/java/com/.../it/tools/AppTest.java
```

At this point, a Git repository has been created locally and it is ready to track the first changes:

```
$ vi src/main/java/com/atreceno/it/tools/App.java
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in
#   working directory)
#
#   modified:   src/main/java/com/atreceno/it/tools/App.java
#
no changes added to commit (use "git add" and/or "git commit -a")
$ git add .
$ git status
# On branch master
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   src/main/java/com/atreceno/it/tools/App.java
#
$ git commit -m 'Add a comment to the main function'
[master 836fb89] Add a comment to the main function
 1 file changed, 1 insertion(+)
```

Git defines three states for a file: *committed*, *modified* and *staged*. Typically, a file that is part of the version control system is in the *committed* state. Then the file is changed so it converts to the *modified* state and finally it is added to the *stage* state so it can be included in the next commit snapshot. This is why even though the file has been modified, it needs to be added with `git add` before committing it. Additionally, it is possible to use the short-cut `git commit -a` to automatically stage files that have been modified or deleted. However, this will not include new files.

During this study, all commit messages were formatted according to the guidelines provided by Git. Here, for brevity, a short version is being used with the parameter `-m`. Popular frameworks such as Spring or NodeJS, offer additional guidelines to contribute to their projects and are usually a good source of best practices, not only to format a commit message but to work within a team.

A.3 Adding a Remote Repository

In order to share the project with other developers, a remote Git repository must first to be added. It is possible to have more than one remote repository and typically they are in different locations. The following listing shows how to use a private server as a remote repository:

```
$ cd ~/Development/git
$ git clone --bare dummy dummy.git
$ scp -r dummy.git agustin@sochi.gast.it.uc3m.es:/opt/git
$ git remote add uc3m \
agustin@sochi.gast.it.uc3m.es:/opt/git/dummy.git
```

To use GitHub, a repository has to first be created in the server. It is very straightforward, but all the details can be found online[3]. Once the repository exists in the server, the code is uploaded by issuing the command `git push` as follows:

```
$ git clone --bare agustin@git.server.com:/opt/git/dummy.git
```

[3] https://help.github.com/articles/create-a-repo

```
$ cd dummy.git
$ git push --mirror https://github.com/atreceno/dummy.git
$ git remote add origin https://github.com/atreceno/dummy.git
```

It is recommended to create a `README.md` file in the home directory of the repository. Files with the extension `.md` use Markdown syntax:

> Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).

GitHub uses a special flavoured Markdown with additional functionality such as fenced code blocks, syntax highlighting and task lists.

Another essential file is `.gitignore`, which contains patterns of files to be ignored by Git. This file depends strongly on the language and IDE being used. A good starting point is to use the `.gitignore` file provided by GitHub when creating a new repository.

A.4 Working within a Team

In the next scenario, there are two team members, Alice and Bob, contributing to the project. To start, they will need to clone the repository to have a local copy of the project:

```
$ git clone alice@git.server.com:dummy.git
$ cd dummy
$ git remote
origin
$ git branch -a
* master
  remotes/origin/master
$ git lola
* 35c28d7 (HEAD, origin/master, origin/HEAD, master) Add a ...
* 35c28d7 (HEAD, origin/master) Add a comment with a timestamp
* 836fb89 Add a comment to the main function
* c04c492 Add scaffolding created by maven
```

Alice will be working on a new feature, while Bob will be solving issues. They create their respective topic branches and commit their changes: (on Alice's computer)

```
$ git branch linearSearch
$ git branch -a
  linearSearch
* master
  remotes/origin/master
$ git checkout linearSearch
Switched to branch 'linearSearch'
$ vi src/main/java/com/atreceno/it/tools/LinearSearch.java
$ vi src/test/java/com/atreceno/it/tools/LinearSearchTest.java
$ git status
# On branch linearSearch
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   src/main/java/com/atreceno/it/tools/LinearSearch.java
#   src/test/java/com/atreceno/it/tools/LinearSearchTest.java
$ git add .
$ git status
# On branch linearSearch
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   src/main/java/.../it/tools/LinearSearch.java
#   new file:   src/test/java/.../it/tools/LinearSearchTest.java
#
```

At this point, the files are in staged state, meaning that this is the snapshot that will go in the next commit. However, Alice forgot to delete one of the TODO comments so she edits the file and saves it. This last version, without the comment, will not go in the next commit unless Alice adds the new snapshot as it is shown in the next listing:

```
$ git status
# On branch linearSearch
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   src/main/java/.../it/tools/LinearSearch.java
#   new file:   src/test/java/.../it/tools/LinearSearchTest.java
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes)
#
#   modified:   src/main/java/.../it/tools/LinearSearch.java
```

```
#
$ git add src/main/java/com/atreceno/it/tools/LinearSearch.java
$ git status
# On branch linearSearch
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   src/main/java/.../it/tools/LinearSearch.java
#   new file:   src/test/java/.../it/tools/LinearSearchTest.java
#
$ git commit
[linearSearch b4f999a] Add linear search algorithm
 2 files changed, 55 insertions(+)
 create mode 100644 src/main/java/.../tools/LinearSearch.java
 create mode 100644 src/test/java/.../tools/LinearSearchTest.java
```

Now she can check the last commit with the `git log` command, like so:

```
$ git log -n1
commit b4f999a45231252e8258b858f760ad485c0dea67
Author: Agustin Treceno <agustin.treceno@gmail.com>
Date:   Fri Jun 28 14:43:29 2013 +0100

    Add linear search algorithm

    In computer science, linear search or sequential search is a
    method for finding a particular value in a list, that consists
    of checking every one of its elements, one at a time and in
    sequence, until the desired one is found.

    More info: http://en.wikipedia.org/wiki/Linear_search
$ git lola
* b4f999a (HEAD, linearSearch) Add linear search algorithm
* 35c28d7 (origin/master, origin/HEAD, master) Add a comment ...
* 836fb89 Add a comment to the main function
* c04c492 Add scaffolding created by maven
```

The commit message uses imperative tense for the subject, and the description has been capped at 72 characters, following the guidelines provided by Git.

At the same time, Bob was working on a issue found in `App.java`. He is about to commit his changes but he first checks that there aren't any trailing spaces with the command `git diff --check`: (on Bob's computer)

```
$ git clone bob@git.server.com:dummy.git
```

```
$ cd dummy
$ vi src/main/java/com/atreceno/it/tools/App.java
$ git diff --check
src/main/java/com/.../it/tools/App.java:8: trailing whitespace.
+    // This is a basic example
$ vi src/main/java/com/atreceno/it/tools/App.java
$ git diff --check
$ git commit -a
[issue13 1b4a5c1] Fix issue13
 1 file changed, 4 insertions(+), 6 deletions(-)
```

Once his changes have been committed, it is time to merge them with their local master branch as he was working on another branch to solve this issue. Afterwards, he can push the new changes up to the server:

```
$ git checkout master
Switched to branch 'master'
$ git merge issue13
Updating 35c28d7..1b4a5c1
Fast-forward
 src/main/java/com/atreceno/it/tools/App.java | 10 ++++------
 1 file changed, 4 insertions(+), 6 deletions(-)
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
nothing to commit (working directory clean)
$ git lola
* 1b4a5c1 (HEAD, master, issue13) Fix issue13
* 35c28d7 (origin/master, origin/HEAD) Add a comment with a ...
* 836fb89 Add a comment to the main function
* c04c492 Add scaffolding created by maven
$ git push origin master
Counting objects: 19, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (10/10), 785 bytes, done.
Total 10 (delta 0), reused 0 (delta 0)
To bob@git.server.com:dummy.git
   35c28d7..1b4a5c1  master -> master
$ git branch -d issue13
Deleted branch issue13 (was 1b4a5c1).
```

Alice wants to share her work with Bob so she creates a new branch in the server. However, before doing that, Alice wants to know if she needs to update her master branch: (on Alice's computer)

```
$ git lola
* b4f999a (HEAD, linearSearch) Add linear search algorithm
* 35c28d7 (origin/master, origin/HEAD, master) Add a comment ...
* 836fb89 Add a comment to the main function
* c04c492 Add scaffolding created by maven
$ git fetch origin
$ git lola
* 1b4a5c1 (origin/master, origin/HEAD) Fix issue13
| * b4f999a (HEAD, linearSearch) Add linear search algorithm
|/
* 35c28d7 (master) Add a comment with a timestamp
* 836fb89 Add a comment to the main function
* c04c492 Add scaffolding created by maven
$ git push origin linearSearch:linear
...
To alice@git.server.com:dummy.git
 * [new branch]      linearSearch -> linear
$ git lola
* 1b4a5c1 (origin/master, origin/HEAD) Fix issue13
| * b4f999a (HEAD, origin/linear, linearSearch) Add linear ...
|/
* 35c28d7 (master) Add a comment with a timestamp
* 836fb89 Add a comment to the main function
* c04c492 Add scaffolding created by maven
```

Bob downloads the new branch, fixes an issue and pushes the new change: (on Bob's computer)

```
$ git co -b linear origin/linear
$ vi src/main/java/com/atreceno/it/tools/LinearSearch.java
$ vi src/test/java/com/atreceno/it/tools/LinearSearchTest.java
$ mvn test
$ git commit -a
[linear 007fe6c] Rename classes and fix equal comparison in loop
 2 files changed, 12 insertions(+), 12 deletions(-)
$ git push origin linear:linear
...
To bob@git.server.com:dummy.git
   b4f999a..007fe6c  linear -> linear
```

Meanwhile, Alice was working on a second feature but now that Bob has finished his job, she wants to push the first feature up to the server: (on Alice's computer)

```
$ git merge origin/linear
Updating b4f999a..007fe6c
```

```
Fast-forward
 src/main/java/com/atreceno/it/tools/LinearSearch.java     |  6
 src/test/java/com/atreceno/it/tools/LinearSearchTest.java | 18
 2 files changed, 12 insertions(+), 12 deletions(-)
$ git co master
Switched to branch 'master'
Your branch is behind 'origin/master' by 1 commit, and can be
fast-forwarded.
$ git merge origin/linear
Updating 35c28d7..007fe6c
Fast-forward
 src/main/java/com/atreceno/it/tools/LinearSearch.java     | 16
 src/test/java/com/atreceno/it/tools/LinearSearchTest.java | 39
 2 files changed, 55 insertions(+)
 create mode 100644 src/main/java/.../tools/LinearSearch.java
 create mode 100644 src/test/java/.../tools/LinearSearchTest.java
$ git push origin master
To alice@git.server.com:dummy.git
 ! [rejected]        master -> master (non-fast-forward)
error:
failed to push some refs to 'alice@git.server.com:dummy.git'
```

Alice is not allowed because Bob pushed his fix to master so she needs to fetch Bob's change to her local master branch:

```
$ git pull
Merge made by the 'recursive' strategy.
 src/main/java/com/atreceno/it/tools/App.java | 10 ++++------
 1 file changed, 4 insertions(+), 6 deletions(-)
$ git push origin master
...
To alice@git.server.com:dummy.git
   1b4a5c1..5697b3f  master -> master
$ git lola
*   5697b3f (HEAD, origin/master, origin/HEAD, master) Merge ...
|\
| * 1b4a5c1 Fix issue13
* | 007fe6c (origin/linear, linearSearch) Rename classes and ...
* | b4f999a Add linear search algorithm
|/
* 35c28d7 Add a comment with a timestamp
* 836fb89 Add a comment to the main function
* c04c492 Add scaffolding created by maven
```

The workflow between Alice and Bob is shown in Figure A.1.

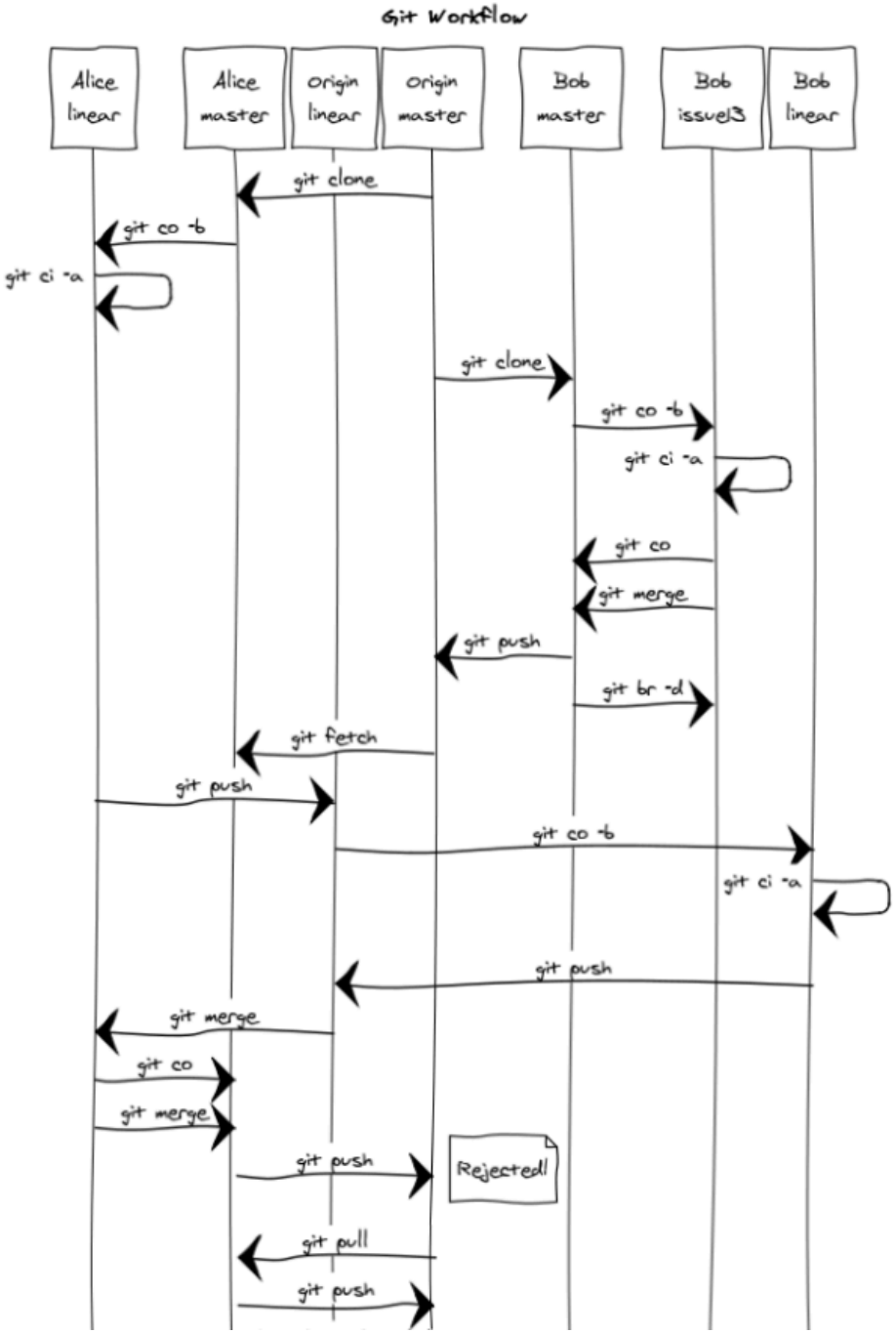

Figure A.1: Git Workflow

To find out more about Git and other use cases, Chacon (2009) explains all the details about this distributed Version Control System (VCS).

Installing MySQL

B.1 Installation on Mac OS X

MySQL comes in several flavours:

- MySQL Enterprise Edition
- MySQL Cluster CGE (Carrier Grade Edition)
- MySQL Community Server
- MySQL Cluster

The first two are offered under a commercial license while the last two are offered under a GNU General Public License (GPL). MySQL Cluster is based on a distributed computing architecture in which each node is self-sufficient, in the sense that they do not share memory or disk storage. It is designed to provide high availability and low latency. Additionally, there are other tools available such as MySQL Workbench, command-line utilities and drivers for different platforms.

MySQL Community Server was used during the implementation of *Diaulos*, the solution architecture based on Java. There are two formats available for the OS X operative system: DMG Archive and Compressed TAR Archive. The former is a proprietary disk image format developed by Apple that, once opened, looks and acts like an actual disk or volume. The latter is a TAR file that has been compressed with `gzip` utility. In this case, MySQL was installed using the DMG file from the command line. After downloading the file from the Internet, it is recommended to verify its integrity. It is straightforward to calculate the MD5 (Message-Digest algorithm 5) checksum of the file and make sure that it matches the one provided on the MySQL download page[1]:

[1] http://dev.mysql.com/downloads/mysql

```
$ md5 -q ~/mysql-5.6.8-rc-osx10.7-x86_64.dmg
11a8421b4b3c7e43175e737a3e786df9
```

It is also good practice to create a specific `mysql` user to own the MySQL directory and data. In Mac OS X 10.7 the user already exists within the system:

```
$ cat /etc/passwd | grep "mysql"
_mysql:*:74:74:MySQL Server:/var/empty:/usr/bin/false
```

The disk image can be mounted with the `hdiutil` command:

```
$ hdiutil attach ~/mysql-5.6.8-rc-osx10.7-x86_64.dmg -quiet
```

The disk image contains four items: the preference pane, the start-up package, the `ReadMe.txt` file and the main installation package:

```
$ ls -1 /Volumes/mysql-5.6.8-rc-osx10.7-x86_64
MySQL.prefPane
MySQLStartupItem.pkg
ReadMe.txt
mysql-5.6.8-rc-osx10.7-x86_64.pkg
```

The first two items enable the server to start and stop as well as the automatic start-up of MySQL when the system boots. The difference is that the preference pane uses the System Preference window and the start-up item package uses the command line. The content of the installation package can be viewed with the command `ls` or `find` because the package is actually a folder:

```
$ cd mysql-5.6.8-rc-osx10.7-x86_64.pkg
$ find . -type f
./Contents/Archive.bom
./Contents/Archive.pax.gz
./Contents/Info.plist
./Contents/PkgInfo
./Contents/Resources/en.lproj/Description.plist
./Contents/Resources/License.txt
./Contents/Resources/package_version
./Contents/Resources/postflight
./Contents/Resources/preflight
./Contents/Resources/ReadMe.txt
```

The first file, `Archive.bom`, contains a list of files to be installed and it is used in the verification and uninstall processes. The content of the file can be viewed

using the Unix command `lsbom`. The extension of the file stands for Bill of Materials. The second file of the list, `Archive.pax.gz`, contains the files to be installed, archived with `pax` and compressed with `gzip`. `Info.plist` and `Description.plist` are XML documents with information about the package such as the name, size and default location. The file `PkgInfo` contains information about the creator of the package. `License.txt` and `ReadMe.txt` are shown to the user when installing the package. The `preflight` script is run before the installation of the package. In this case, it just renames the folder `/usr/local/mysql` to `/usr/local/mysql.bak`, if it already exists. In a similar way, `postflight` is run after the installation. The script creates the MySQL system tables by executing `scripts/mysql_install_db` and changing the owner of the `data` folder to the user `mysql`.

To install both, the main installation package and the start-up item, the command `installer` is used:

```
$ sudo installer -pkg mysql-5.6.8-rc-osx10.7-x86_64.pkg -tgt /
installer: Package name is MySQL 5.6.8-rc-community for Mac OS X
installer: Installing at base path /
installer: The install was successful.
$ sudo installer -pkg MySQLStartupItem.pkg -tgt /
installer: Package name is MySQL Startup Item
installer: Installing at base path /
installer: The install was successful.
```

When a new package is installed, a new entry with information relative to the package is added to the file `InstallHistory.plist` as shown below:

```
$ grep -B 1 "MySQL" /Library/Receipts/InstallHistory.plist
<key>displayName</key>
<string>MySQL 5.6.8-rc-community for Mac OS X</string>
--
<key>displayName</key>
<string>MySQL Startup Item</string>
```

The package `MySQLStartupItem.pkg` adds the variable `MYSQLCOM=-YES-` to the system configuration file `/etc/hostconfig`, enabling the automatic start-up of the server when the system reboots. To disable this functionality, just change the variable to `MYSQLCOM=-NO-`. Additionally, the package provides the script `/Library/StartupItems/MySQLCOM/MySQLCOM` to start and stop the server. This is actually a wrapper of the `mysql.server` init script located in `/usr/local/mysql/support-files`.

Once MySQL Community Server has been installed, the installation folder `/usr/local/mysql` should have the content shown in Table B.1.

Directory	Contents of Directory
bin	Client and server programs
data	Logs and databases
docs	Complete manual in info format
include	Header .h files
lib	Libraries
man	Unix manual pages
mysql-test	Test suite for the MySQL daemon
scripts	Post-install script
share	Error messages
sql-bench	Benchmarks
support-files	Sample configuration files

Table B.1: Content of MySQL home folder

B.2 Post-installation set-up

There are some post-installation tasks that need to be addressed. Most of these tasks are included in the script `mysql_install_db`, such as the creation of the system tables. In more detail, it creates the `mysql` and `test` databases under the `data` directory. The script also creates accounts for `root` and `anonymous` users but it does not assign passwords to them or restrict access to the databases.

There are several ways to secure a database. One of them is to invoke the script `mysql_install_db` with the `--random-passwords` option as recommended in the MySQL Reference Manual. Other ways of assigning passwords include the use of the command `mysqladmin` or the statement `set password`. In this case, this script was already executed by the post-installation process without that option but, in a fresh installation of MySQL, the folder `data` could have been deleted and then populated again by running the script. Please note, that it would be necessary to assign the right permissions and owner back to the `data` folder and sub-folders. Unfortunately, the script `mysql_install_db` provided by MySQL uses the command `tr` to generate the passwords, which is affected by locale settings, as shown below:

```
$ cat /dev/urandom | tr -dc "[:alnum:]" | fold -w 8  | head -1
```

```
tr: Illegal byte sequence
$ LC_ALL=C
$ cat /dev/urandom | tr -dc "[:alnum:]" | fold -w 8  | head -1
RyjSQn9k
```

This issue can be solved by changing all locale variables temporarily to `ANSI C` or adding `LC_ALL=C` to the original script to make it work:

```
$ sudo mv data data_notsecure
$ sudo ./scripts/mysql_install_db --random-passwords
Installing MySQL system tables...
A RANDOM PASSWORD HAS BEEN SET FOR THE MySQL root USER !
You will find that password in '/Users/agustin/.mysql_secret'.
Also, the account for the anonymous user has been removed.
$ sudo chmod 755 data
$ sudo chmod 755 data/mysql
$ sudo chmod 755 data/test
$ sudo chown -R mysql data
```

After populating the system tables with the `--random-passwords` option, the password for the MySQL `root` user must be changed. The only command that will be accepted at this point is `set password` without any parameters:

```
$ sudo ./bin/mysqld_safe
$ bin/mysql -u root -p
mysql> set password = password('LBKScxxu');
```

The function `password(str)` calculates and returns the hashed string of the argument. The hashing method depends on the value of the `old_password` system variable. Looking at the process with the command `ps`, it can be seen how the script `mysqld_safe` adds the `--user` option before calling the actual daemon process `mysqld`. Another way of starting the server is using the script `mysql.server` in folder `support-files` which is actually a wrapper of `mysqld_safe`. Once the `root` password has been changed, it is possible to change the password for the rest of the accounts as shown below:

```
mysql> show databases;
mysql> use mysql;
mysql> show tables;
mysql> desc user;
mysql> select Host, User, Password, password_expired from user;
+--------------+------+-------------------------------------------+
| Host         | User | Password                  | password_expired |
+--------------+------+-------------------------------------------+
```

```
| localhost    | root | *E61AD59415FAC0B... | N                |
| jupiter.home | root | *13ACD43B6A3BADB... | Y                |
| 127.0.0.1    | root | *13ACD43B6A3BADB... | Y                |
| ::1          | root | *13ACD43B6A3BADB... | Y                |
+--------------+------+-----------------------------------------+
mysql> set password for 'root'@'jupiter.local' =
    -> password('LBKScxxu');
mysql> set password for 'root'@'127.0.0.1' = password('LBKScxxu');
mysql> set password for 'root'@'::1' = password('LBKScxxu');
```

The `db` table contains rows that permit any anonymous account to access the `test` database and other databases starting with the prefix `test_`. The script `mysql_install_db` deleted the anonymous accounts from the user table so only authorized accounts will now have access to the test databases:

```
mysql> desc db;
mysql> select Host, Db, User from db;
+------+---------+------+
| Host | Db      | User |
+------+---------+------+
| %    | test    |      |
| %    | test\_% |      |
+------+---------+------+
```

If this would not have been a fresh installation and there would have been important information in the folder `data`, there is an alternative way of securing the database by invoking the Perl script `mysql_secure_installation` located in the folder `bin` of the distribution. The script allows for a password change for the MySQL `root` account and the removal of the anonymous account and test databases.

In addition, the script `mysql_install_db` creates a configuration file, `my.cnf`, in the mysql installation folder. The file contains commonly used options so that they don't need to be entered on the command line. The options in the configuration file are grouped by tags that match the name of the program. `my.cnf` is generated from the template `my-default.cnf` located in `{MYSQL_HOME}/support-files` folder. The template replaces the former configuration files `my-small.cnf`, `my-medium.cnf`, etc., that were distributed in previous versions of MySQL. For more information about this and other changes, refer to the release notes: Changes in MySQL 5.6.8. To see the default options that a server will use, issue the command `mysqld --verbose --help` or if the instance is already running, `mysqladmin variables`. The information generated is quite long so it might help to redirect the output to another file and then use `vim` editor with the option `set nowrap`:

```
$ mysqladmin variables -u root -p > /tmp/vars.txt
```

C

Installing Apache Benchmark

Apache HTTP comes with a tool for benchmarking the server, `ab`:

```
$ ab -V
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd.
Licensed to The Apache Software Foundation
```

It is designed to give an idea of how an Apache installation performs. Unfortunately, the revision included in Apache 2.2, shipped by default in Mac OS X 10.7 (Lion), has a bug as shown below:

```
$ ab -n 100 -c 10 http://localhost:8080/diaulos/login
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd.
Licensed to The Apache Software Foundation

Benchmarking localhost (be patient)...apr_socket_recv: Connection
reset by peer (54)
```

To solve this problem, the latest version of Apache HTTP has to be compiled and installed. Before installing Apache HTTP, the PCRE (Perl Compatible Regular Expression) library has to first be installed. This library contains a set of functions that implement regular expressions, which use the same syntax and semantics as Perl 5. The installation of both, PRCE library and Apache HTTP is shown below:

```
$ curl -O http://sourceforge.net/projects/pcre/files/pcre/8.32/
```

```
pcre-8.32.tar.gz
$ tar -zxf pcre-8.32.tar.gz; cd pcre-8.32
$ ./configure --prefix=~/Dev/pcre-8.32/build
$ make install
$ curl -O http://mirrors.ukfast.co.uk/sites/ftp.apache.org/
httpd/httpd-2.4.4.tar.gz
$ tar -zxf httpd-2.4.4.tar.gz; cd httpd-2.4.4
$ ./configure --with-pcre=~/Dev/pcre-8.32/build/bin/pcre-config
$ make
$ sudo cp support/ab /usr/sbin
```

Once installed, the command `ab` works as expected:

```
$ ./ab -n 100 -c 10 -v 0 http://localhost:8080/diaulos/login
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd.
Licensed to The Apache Software Foundation

Benchmarking localhost (be patient).....done
Server Software:        Apache-Coyote/1.1
Server Hostname:        localhost
Server Port:            8080

Document Path:          /diaulos/login
Document Length:        4024 bytes

Concurrency Level:      10
Time taken for tests:   1.537 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      429500 bytes
HTML transferred:       402400 bytes
Requests per second:    65.05 [#/sec] (mean)
Time per request:       153.735 [ms] (mean)
Time per request:       15.374 [ms] (mean, across all concurrent
requests)
Transfer rate:          272.83 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   2.4      0      15
Processing:    15  110 110.4     82     774
Waiting:       15   94  84.1     69     612
Total:         16  111 110.3     83     774
```

```
Percentage of the requests served within a certain time (ms)
  50%     83
  66%    117
  75%    134
  80%    155
  90%    242
  95%    283
  98%    612
  99%    774
 100%    774 (longest request)
```

In this case, the web server (VMware vFabric tc Server) is able to handle 100 requests (10 concurrent) at 65 requests per second.

D

Quote

Agile was used throughout this study. Agile is based on iterative and incremental developments called Sprints. It favours working software, customer collaboration, and responding to change (see Ratcliffe and McNeill 2011). Figure D.1 shows a snapshot of the task board during the early stages of the *Diaulos* project.

It took 15 months for this study to complete, dedicating 8 hours a day. The quote is shown in table D.1.

Description	Quantity	Unit Price	Cost
Infrastructure			
Office rental (monthly)	15	£1,200.00	£18,000.00
Hardware			
MacBook Pro 15-inch	1	£2,200.00	£2,200.00
Server at gast.it.uc3m.es	1	£0.00	£0.00
Software			
Amazon RDS	1	£0.00	£0.00
Apache Benchmark 2.4.4	1	£0.00	£0.00
Firefox Aurora 26.0	1	£0.00	£0.00
GitHub Service	1	£0.00	£0.00

Gliffy Diagrams SaaS	1	£0.00	£0.00
Google Chrome 31.0	1	£0.00	£0.00
Google Image Charts SaaS	1	£0.00	£0.00
Heroku PaaS	1	£0.00	£0.00
Java frameworks and libraries	1	£0.00	£0.00
JavaScript frameworks and libraries	1	£0.00	£0.00
Latex 2.5	1	£0.00	£0.00
Mac OS X 10.7.5	1	£0.00	£0.00
MacVim (snapshot 66)	1	£0.00	£0.00
MongoDB 2.2.3	1	£0.00	£0.00
Mongolab DBaaS	1	£0.00	£0.00
MySQL 5.6.8	1	£0.00	£0.00
MySQL Workbench 5.6.8	1	£0.00	£0.00
New Relic Monitoring	1	£0.00	£0.00
Nodejitsu PaaS	1	£0.00	£0.00
Pandoc 1.11.1	1	£0.00	£0.00
Pivotal CF PaaS	1	£0.00	£0.00
Safari 6.1	1	£0.00	£0.00
Scrumwise SaaS (free 30-day trial)	1	£0.00	£0.00
Siege 3.0.4	1	£0.00	£0.00
STS IDE 3.4	1	£0.00	£0.00
Travis CI	1	£0.00	£0.00
WebStorm 7 (free 30-day trial)	1	£0.00	£0.00
Contractor			
Software Engineer (monthly)	15	£4,000.00	£60,000.00
		Subtotal	£80,200.00
	Tax	21.00%	£16,842.00
		Total	**£97,042.00**

Table D.1: Quote

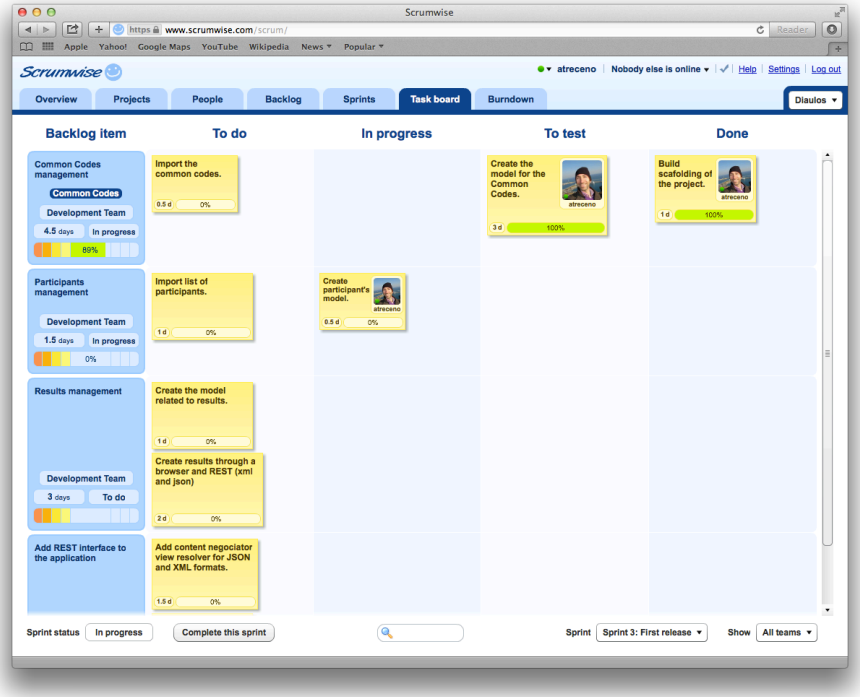

Figure D.1: Scrum task board

A more detail tasks with the duration in number of weeks is shown in table D.2.

Task	Weeks
State-of-the-art	10
Analyse the current situation	2
Download and process Olympic data	2
Agile methodology	2
Setup development environment	2
Git	2
Spring Solution Architecture	46
Java Language	5
Spring Framework	5
Spring Roo	4
Spring Integration	3
Spring JPA	2
Spring Security	2
Hibernate	5
MySQL	2
Create scaffolding	1
Persistence layer	2
Service layer	1
Web layer	2
Deploy to Pivotal CF	1
Deploy to Heroku	1
Amazon RDS	1
Logic Unit Tests	1
Integration Unit Tests	1
Front-end Unit Tests	2
Integration Tests	1
Load Tests	2

Stress Tests	2
NodeJS Solution Architecture	48
JavaScript	5
NodeJS	5
AngularJS	5
Express	2
Jade	1
Socket.io	1
Mongoose	1
D3.js	2
Foundation 4	2
Twitter Bootstrap	2
MongoDB	3
Create scaffolding	3
Implement front-end	3
Implement back-end	2
Deploy to Nodejitsu	1
Deploy to Heroku	1
Mongolab DBaaS	1
Unit tests	2
End-to-end tests	2
Load Tests	2
Stress Tests	2
Documentation	13
Pandoc	1
Latex	1
Create template	1
Write documentation	10

Table D.2: Duration of tasks

Before closing this section, here are some statistics about this study:

Application	Files	Lines	Commits	Online ref.
Diaulos	498	22454	80	177
Stadion	58	4948	47	125
Total	556	27402	127	302

Table D.3: Statistics about this study

References

Andrews, Scott. 2013. "Webinar: Architecture of a modern web app." Video file. http://www.youtube.com/watch?v=oYohGSJStyk.

Ben Alex, Luke Taylor. 2013. "Spring Security Reference Documentation 3.2.0." http://projects.spring.io/spring-security/.

Chacon, Scott. 2009. *Pro git*. Apress.

Clark, Richard, Oli Studholme, Christopher Murphy, and Divya Manian. 2012. *Beginning HTML5 and CSS3*. Apress.

Deinum, Marten, Koen Serneels, Colin Yates, Seth Ladd, and Christophe Vanfleteren. 2012. *Pro Spring MVC: With Web Flow*. Apress.

Fisher, Paul Tepper, and Brian D. Murphy. 2010. *Spring persistence with Hibernate*. Apress.

Flanagan, David. 2012. *JavaScript pocket reference*. O'Reilly.

Flanagan, David, and Mike Loukides. 2011. *JavaScript: the definitive guide*. O'Reilly.

Fowler, Martin. 2003. *Patterns of enterprise application architecture*. Boston: Addison-Wesley.

Friesen, Jeff. 2011. *Beginning Java 7. The expert's voice in Java*. New York, NY: Apress.

Gamma, Erich. 1995. *Design patterns: elements of reusable object-oriented software*. Reading, Mass.: Addison-Wesley.

Garcia-Molina, Hector, Jeffrey D. Ullman, and Jennifer Widom. 2008. *Database Systems: The Complete Book*. 2nd ed.. Upper Saddle River, NJ, USA: Prentice Hall Press.

Gierke, Oliver, Jonathan Brisbin, Thomas Risberg, and Michael Hunger. 2012. *Spring Data*. O'Reilly.

Goncalves, Antonio. 2013. *Beginning Java EE 7*. Apress.

Guckenheimer, Sam, and Neno Loje. 2012. *Agile Software Engineering with Visual Studio: From Concept to Continuous Delivery*. Addison-Wesley Professional.

Hernandez, Michael J. 2003. *Database design for mere mortals: a hands-on guide to relational database design*. Addison-Wesley Professional.

Highsmith, Jim. 2009. *Agile project management: creating innovative products*. Pearson Education.

Ho, Clarence, and Rob Harrop. 2012. *Pro Spring 3. The expert's voice in Spring*. Apress.

Johnson, Rod. 2003. *Expert one-on-one J2EE design and development*. Indianapolis, IN: Wrox.

Juneau, Josh. 2013. *Java EE 7 Recipes: A Problem-Solution Approach*. Apress.

Kainulainen, Petri. 2012. *Spring Data*. Packt Publishing Ltd.

Konda, Madhusudhan. 2012. *Just Spring data access*. Sebastopol, CA: O'Reilly.

Laddad, Ramnivas. 2009. *AspectJ in Action: Enterprise AOP with Spring Applications*. 2nd ed.. Greenwich, CT, USA: Manning Publications Co.

Linwood, Jeff, and Dave Minter. 2010. *Beginning Hibernate. Expert's voice in Java technology*. 2nd ed.. Apress.

Lui, Mark. 2011. *Pro Spring integration. The Expert's voice in Spring*. New York, NY: Apress.

Maher, Jacqui. 2013. "London Calling: Winning the Data Olympics." http://bit.ly/1eMez3o.

Mak, Gary, and Srinivas Guruzu. 2010. *Hibernate Recipes: A Problem-Solution Approach*. Apress.

Millman, Rene. 2012. "IaaS providers: How to select the right company for your cloud." http://bit.ly/NcfTiF.

Powers, Shelley. 2012. *Learning Node*. O'Reilly Media, Inc.

Prasanna, Dhanji R. 2009. *Dependency injection*. Manning Publications Co.

Ratcliffe, Lindsay, and Marc McNeill. 2011. *Agile Experience Design*. New Riders.

Resig, John, and Bear Bibeault. 2013. *Secrets of the JavaScript Ninja*. Manning.

Rimple, Ken, Srini Penchikala, and Ben Alex. 2012. *Spring Roo in action*. Shelter Island, NY: Manning.

Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop. 2013. "Spring Framework Reference Documentation 3.2.5." http://projects.spring.io/spring-framework/.

Rogge, Jacques. 2012. "Was London 2012 the best Olympics ever?" http://bit.ly/HYagnf.

Rubin, Kenneth S. 2012. *Essential Scrum*. Addison-Wesley.

Scarioni, Carlo. 2013. *Pro Spring Security*. 1st ed.. Berkely, CA, USA: Apress.

Stephens, Matt, and Doug Rosenberg. 2010. *Design Driven Testing: Test Smarter, Not Harder*. Apress.

Tahchiev, Petar, and Vincent Massol. 2011. *JUnit in action*. 2nd ed.. Greenwich: Manning.

Walls, Craig. 2011. *Spring in action*. 3rd ed.. Shelter Island: Manning.

Wang, Vanessa, Frank Salim, and Peter Moskovits. 2012. *The Definitive Guide to HTML5 WebSocket*. Apress.

Wilson, Mike. 2012. *Building Node Applications with MongoDB and Backbone*. O'Reilly.