Instituto Superior de Engenharia do Porto

RealCrono – sistema de difusão de resultados em tempo real na *web*.

Jorge Manuel Nunes Carneiro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Arquiteturas, Sistemas e Redes

Orientador: Paulo Gandra de Sousa

Júri:

Presidente:

Doutor Luis Miguel Moreira Lino Ferreira, DEI/ISEP

Vogais:

Doutor Luis Miguel Pinho Nogueira, DEI/ISEP

Doutor Paulo Alexandre Gandra de Sousa, DEI/ISEP

Porto, Outubro 2013

Dedico este trabalho a ti, Ana...

Resumo

A velocidade de difusão de conteúdos numa plataforma *web*, assume uma elevada relevância em serviços onde a informação se pretende atualizada e em tempo real. Este projeto de Mestrado, apresenta uma abordagem de um sistema distribuído de recolher e difundir resultados em tempo real entre várias plataformas, nomeadamente sistemas móveis. Neste contexto, tempo real entende-se como uma diferença de tempo nula entre a recolha e difusão, ignorando fatores que não podem ser controlados pelo sistema, como latência de comunicação e tempo de processamento.

Este projeto tem como base uma arquitetura existente de processamento e publicação de resultados desportivos, que apresentava alguns problemas relacionados com escalabilidade, segurança, tempos de entrega de resultados longos e sem integração com outras plataformas.

Ao longo deste trabalho procurou-se investigar fatores que condicionassem a escalabilidade de uma aplicação *web* dando ênfase à implementação de uma solução baseada em replicação e escalabilidade horizontal. Procurou-se também apresentar uma solução de interoperabilidade entre sistemas e plataformas heterogêneas, mantendo sempre elevados níveis de performance e promovendo a introdução de plataformas móveis no sistema. De várias abordagens existentes para comunicação em tempo real sobre uma plataforma *web*, adotou-se um implementação baseada em WebSocket que elimina o tempo desperdiçado entre a recolha de informação e sua difusão.

Neste projeto é descrito o processo de implementação da API de recolha de dados (Collector), da biblioteca de comunicação com o Collector, da aplicação web (Publisher) e sua API, da biblioteca de comunicação com o Publisher e por fim a implementação da aplicação móvel multi-plataforma.

Com os componentes criados, avaliaram-se os resultados obtidos com a nova arquitetura de forma a aferir a escalabilidade e performance da solução criada e sua adaptação ao sistema existente.

Palavras-chave: Escalabilidade, Interoperabilidade, Comunicação em tempo real, WebSocket's, Plataformas móveis.

Abstract

The rate of diffusion of content in a web platform takes on an increased relevance in services where information is to be updated in real time. This Master's project presents an approach for an architecture of a distributed system to collect and disseminate results in real time across multiple platforms, including mobile systems. In this context, real time means zero delay between collection and dissemination, ignoring factors that cannot be controlled by the system, such as communication delay and processing time.

This project is based on an existing architecture for processing and publishing sports results which had some problems related to scalability, security, delivery times for long results, and a lack of integration with other platforms.

Throughout this study it was sought to investigate factors that constrained the scalability of a web application, giving emphasis to the implementation of a scalable solution based on replication and horizontal scalability. It was also sought to provide a solution for interoperability between heterogeneous platforms and systems, while maintaining high levels of performance and promoting the introduction of mobile platforms in the system. From several existing approaches for real-time communication over a web platform, there was developed an implementation based in WebSocket which eliminates wasted time between data collection and dissemination.

This project describes the process of implementation of the API for data collection (Collector), the library of communication with the Collector, the web application (Publisher) and its API, the library for communicating with the Publisher, and, finally, the implementation of the multi-platform mobile application.

With the components created, the results obtained with the new architecture were evaluated in order to assess the scalability and performance of the solution set and its adaptation to the existing system .

Keywords: Scalability, Interoperability, Real-time communication, WebSocket's, Mobile platforms .

Agradecimentos

Na realização deste projeto de Mestrado foram várias as pessoas que me ajudaram e incentivaram, sem as quais este trabalho teria sido ainda mais difícil. Para algumas delas, pelo especial apoio que me prestaram ao longo desta jornada quero agradecer de uma forma particular, não podendo deixar de expressar a minha gratidão.

A minha esposa, que me apoiou e motivou na conclusão deste projeto, estou certo que sem ela este trabalho não seria possível.

À minha família, que sempre estiveram onde e quando mais precisei.

Ao Engenheiro Paulo Pinto, responsável pela empresa RP2 – Processamento e Comunicação Lda (RP2), pela oportunidade que me deu de tornar este projeto possível.

Ao Doutor Paulo Grandra pela orientação deste trabalho, discussão da arquitetura e elaboração deste documento.

Aos meus colegas de Mestrado que de uma forma indireta sempre me ajudaram neste projeto.

A todos, o meu Muito Obrigado.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

SOA	*Service-oriented architecture*
SOAP	*Simple Object Access Protocol*
REST	*Representational State Transfer*
DBMS	*Database management system*
RIA	*Rich Internet application*
API	*Application programming interface*
ODBC	*Open Database Connectivity*
GSM	*Global System for Mobile*
URL	*Uniform Resource Locator*
JSON	*JavaScript Object Notation*
SEO	*Search Engine Optimization*
DOM	*Document Object Model*
HTTP	*Hypertext Transfer Protocol*
XHR	*XMLHttpRequest*
WWW	*World Wide Webl*
XML	*eXtensible Markup Language*
DNS	*Domain Name System*
CDN	*Content Delivery Network*
AMQP	*Advanced Message Queuing Protocol*
YAML	*Yet Another Markup Language*
AWS	*Amazon Web Services*
P2P	*Peer-to-peer*
URI	*Uniform Resource Identifier*

1 Introdução

Durante os últimos anos tem-se vindo a assistir a uma constante evolução nas tecnologias Web. A internet deixou de ser apenas um repositório de ficheiros estáticos e passou a disponibilizar conteúdos dinâmicos e interativos, assumiu um papel importante na interoperabilidade entre sistemas e também um meio de difusão de informação privilegiado, permitindo a difusão de conteúdos para um elevado número de dispositivos. A velocidade em que os conteúdos são difundidos numa plataforma *web*, assume uma elevada relevância em serviços onde a informação se pretende atualizada e em tempo real. Serviços noticiosos, jogos on-line, bolsa de valores entre muitos outros carecem desta necessidade.

Este projeto de Mestrado, apresenta um caso prático, de uma necessidade de difusão em tempo real através de uma plataforma *web*. O conceito "tempo real" aqui deve ser entendido como tempo nulo entre a produção e visualização de um determinado conteúdo, ignorando questões relacionadas com latência de comunicação ou mesmo tempo gasto em processamento de dados. Pretende-se assim a implementação de um sistema distribuído que recolha informação de cronometragem de uma prova desportiva, a envie para processamento e por fim a difunda na web em tempo real.

Este projeto surge de uma arquitetura existente que se pretende substituir, por uma solução escalável com elevados níveis de performance, que permita a interoperabilidade entre outros sistemas e que difunda os resultados em tempo real. Pretende-se também a implementação de um aplicação móvel que permita usufruir das funcionalidades destes dispositivos, dando ênfase à comunicação em tempo real, aumentando assim a visibilidade do serviço para um público mais alargado.

1.1 Contextualização

Este trabalho foi efetuado no contexto do projeto de Mestrado em Engenharia Informática na área de Arquitetura, Sistemas e Redes no Instituto Superior de Engenharia do Porto.

O projeto exposto neste documento foi desenvolvido na empresa RP2 - Processamento e Comunicações Lda, que desde a sua fundação se tem empenhado no desenvolvimento de soluções em diversas áreas de processamento e comunicação de dados. Uma das áreas onde se encontra envolvida é no processamento e divulgação de resultados de provas desportivas. É nesta área que este projeto de Mestrado foi realizado, procurando encontrar soluções aos problemas colocados.

Revendo um pouco a história, a primeira solução desenvolvida pela RP2 remonta ao ano de 2001, altura onde quase todas as áreas da sociedade (Educação, Saúde, Desporto, etc.) estavam a ser adaptadas a sistemas de informação. Os sistemas de informação eram uma realidade, a internet ainda era um serviço em expansão e os sistemas móveis apenas serviam para comunicação de voz ou mensagens de texto (GSM 2G). Nessa altura surge o desafio de implementar uma solução que permitisse informatizar cronometragens desportivas, concretamente provas de *rally* de uma forma centralizada. Os sistemas existentes eram escassos, caros e não se adaptavam aos regulamentos nacionais. A solução foi implementada e em poucos anos o sistema calculou centenas de classificações oficiais desde provas regionais a nacionais.

Em 2005 surge a necessidade destas classificações serem disponibilizadas na Internet. O serviço tornou-se uma referência nacional na área de cronometragens em provas de *rally* pela sua forte presença na internet e em 2008 surge a oportunidade de tornar o sistema multidisciplinar e distribuído de forma a que o processo de recolha e processamento fosse integrado. Desde então realizaram-se provas de todo-o-terreno de automóveis e motos, circuitos, montanha e enduro, provas pontuáveis para os mais diversos campeonatos desde nacionais a internacionais.

A solução que se pretende para este projeto de Mestrado é dar continuidade a um serviço existente desde o ano de 2001 que tem vindo a sofrer um conjunto de adaptações a novos requisitos e realidades, sofrendo com isso alguns impactos na sua arquitetura inicial, tendo criado um conjunto de problemas de segurança, performance e escalabilidade do sistema para outro tipo de soluções. Pretende-se melhorar a arquitetura existente e os serviços disponíveis na web torná-los mais céleres de forma a dar a sensação de "tempo real". Por fim disponibilizar os serviços em plataformas móveis, devido ao crescimento desse mercado.

1.2 Problema

Ao longo da última década o sistema existente tem sofrido um conjunto de alterações devido a novos requisitos. Estas alterações devido a um fraco planeamento ou falta de previsão futura tem deixado algumas fragilidades na sua arquitetura, levantado um conjunto de problemas de segurança, performance e escalabilidade.

A figura 1 é uma representação de alto nível da arquitetura, onde se demonstra a distribuição de componentes e intervenientes no sistema, mostra também o problema de latência existente entre a recolha de informação e a entrega de resultados ao público.

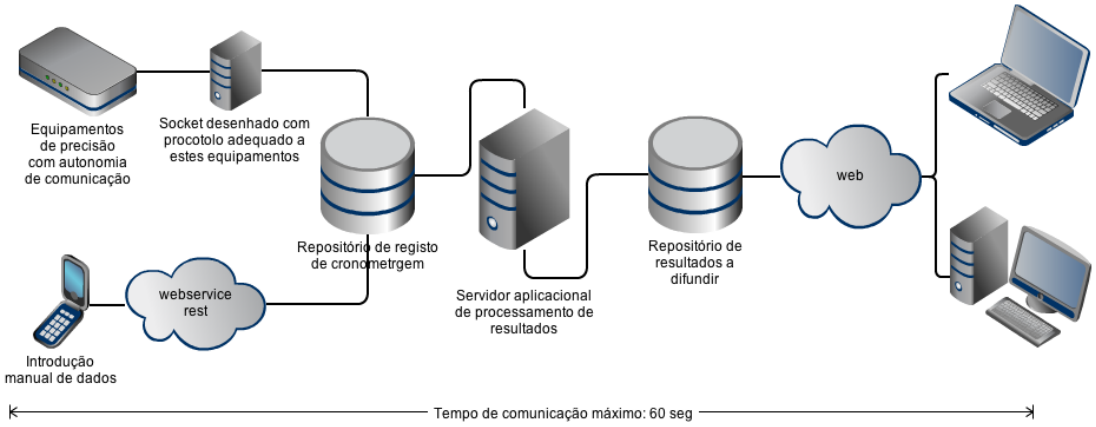


Figura 1 - Arquitetura existente

O servidor aplicacional é o componente central de todo o sistema, responsável pela interoperabilidade entre a recolha e divulgação de resultados. Numa fase inicial tratava-se de um componente isolado onde apenas efetuava os cálculos de resultados. Com o evoluir dos anos ganhou outra relevância, com serviços de difusão na Web e mais recentemente com a recolha de informação diretamente dos equipamentos ou manualmente através de equipamentos móveis. No entanto, o tempo de comunicação entre a recolha de resultados, processamento e divulgação é considerado elevado, chegando a ser superior a **60 segundos**.

Segue-se uma diagrama de sequência que descreve os elementos mais relevantes nesta arquitetura e como comunicam entre si.

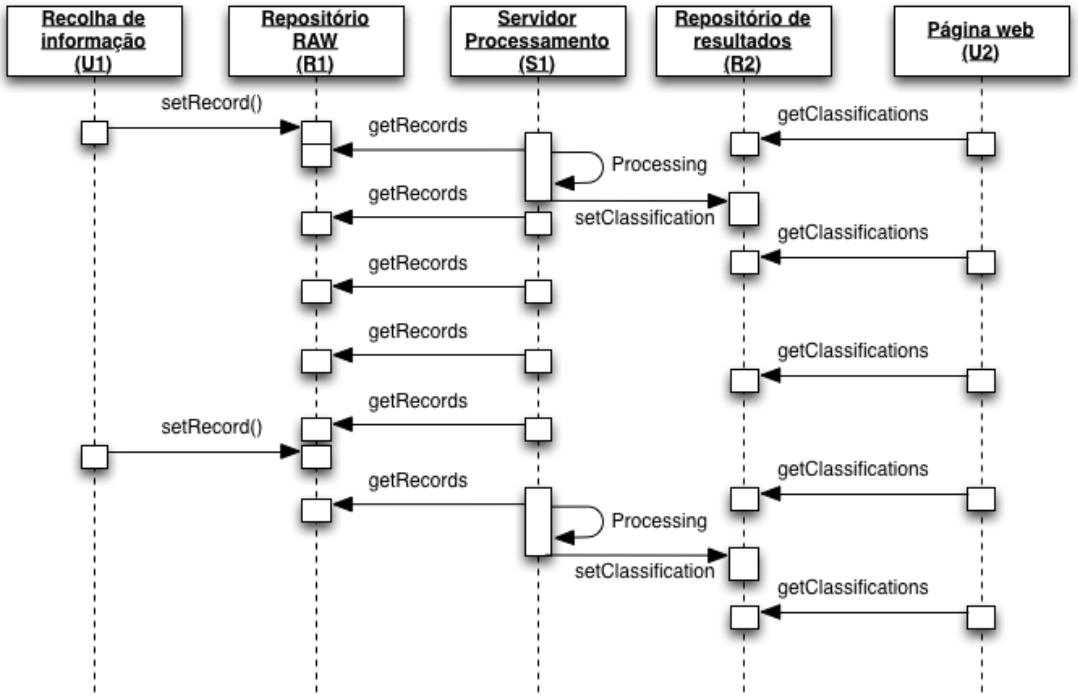


Figura 2 - Diagrama de sequência do sistema existente.

São vários os dispositivos de recolha de informação (U1), estes encontram-se no local da prova e são responsáveis por recolher a informação necessária ao processamento de resultados. Esses locais tanto podem ser no centro de uma cidade como no alto de uma montanha em condições climatéricas completamente adversas, razão pela qual nem sempre as condições de comunicação são as ideais. Nesse sentido, existe um conjunto de equipamentos com meios de comunicação diferentes que ajudam a ultrapassar algumas dessas dificuldades de forma a garantir que independentemente do meio e da velocidade a informação recolhida será enviada através da internet para (R1).

Existem *webservices* que armazenam num repositório comum (repositório RAW) (R1), informação ainda por tratar. Encontram-se localizados num *datacenter* dois *webservices*, um SOAP e outro REST, que aguardam dados dos vários dispositivos (U1). Existe também um serviço que disponibiliza um *socket* à escuta num porto com o mesmo propósito dos *webservices,* mas para aplicações mais antigas que ainda não foram adaptadas a um dos *webservices*. O objetivo destes serviços é receber e guardar num repositório os dados ainda por tratar.

O servidor de processamento de dados (S1), encontra-se num local normalmente próximo da prova conhecido por secretariado ou centro de processamento. Trata-se do serviço responsável pela gestão e processamento de informação onde periodicamente consulta R1 a

verificar se existe nova informação, recolhendo-a e processando-a para posteriormente enviar a R2.

O repositório de resultados (R2), localiza-se num *datacenter* de forma a garantir total disponibilidade do serviço, armazenando os dados processados.

O servidor web (U2), recorrendo aos dados armazenados em R2, disponibiliza uma interface *web* onde podem ser consultados os resultados das provas efetuadas.

A separação dos vários elementos esteve relacionada com o facto das operações serem executadas em locais diferentes, alguns em constante mobilidade, como é o caso de U1 e S1. Este facto foi a razão para que os componentes comunicassem de uma forma isolada e assíncrona, de forma a garantir independência entre eles e garantindo que caso um dos componentes falhe, parte do sistema se mantenha operacional.

Embora esta abordagem fosse a mais adequada e segura no momento da implementação, tem-se detetado alguns problemas; tempos de entrega de informação longos e com um grande *overload* de comunicação nos repositórios de dados.

O serviço tem uma utilização sazonal, tendo alguns picos de utilização que já provocaram falhas no serviço, nomeadamente em eventos com o número de acessos muito elevados.

Ao nível da segurança, parte do sistema comunica diretamente com as bases de dados remotas que se encontram expostas em servidores de acesso público, como é o caso de S1 com R1 e R2, usando apenas as credenciais de autenticação do DBMS em ligação ODBC.

1.3 Objetivos

Pretende-se com este projeto de Mestrado :

- Melhorar uma arquitetura existente que fruto da evolução tecnológica tende em convergir para serviços num modelo de "*cloud computing*".
- Procurou-se também descrever o conceito de interoperabilidade através do desenvolvimento de uma arquitetura baseada em SOA combinada com o uso de *Web Services*, *socket* e WebSocket, com o objetivo de promover a integração entre dispositivos de tecnologias diferentes, enfatizando a introdução dos *smartphones* num sistema de informação existente.
- Melhorar todo o processo de comunicação entre componentes, de forma a que o tempo de divulgação da informação seja mitigado.
- Permitir apenas que as comunicações entre os vários componentes sejam efetuadas através de serviços.
- Desenvolver serviços que permitam ser consumidos por serviços externos, nomeadamente plataformas móveis.

- Desenvolvimento de protótipos de plataformas móveis de forma a integrar com a nova arquitetura.
- Implementar uma solução escalável com elevados níveis de performance que permita colmatar os picos de utilização sazonais.

1.4 Metodologia

A metodologia utilizada começa com a revisão do estado de arte, investigando sobre as temática de sistemas escaláveis, comunicação em "tempo real" sobre internet, serviços SOA e plataformas móveis. O estudo consiste numa análise das diversas tecnologias e soluções já implementadas em diversas situações onde o atual problema pode surgir, particularmente em sistemas escaláveis e comunicação em "tempo real" na internet.

Tendo por base algumas especificações prévias, pretendeu-se projetar uma nova arquitetura padronizada sobre um sistema modular. O projeto começa por estudar um conjunto de tecnologias que face às especificações propostas seriam relevantes na implementação, nomeadamente: desenvolvimento de sistemas escaláveis; tecnologias e técnicas existentes de comunicação em tempo real sobre a internet dando foco à tecnologia de WebSocket's em HTML5; desenvolvimento de aplicações para plataformas móveis.

A fase de implementação da nova arquitetura, em parte, foi desenvolvida paralelamente com a fase de investigação devido à extensão de componentes e diversidade de temáticas abordadas neste projeto. Por fim, de forma a confirmar os objetivos propostos foram efetuados um conjunto de testes de forma a verificar e validar a nova arquitetura face à antiga.

1.5 Organização da dissertação

Este projecto de Mestrado encontra-se organizado em 6 capítulos.

No Capítulo 2 apresenta-se um estudo sobre o estado da arte, técnicas e soluções de escalabilidade, interoperablidade e comunicação em tempo real na internet, apresentam-se ainda alguns estudos comparativos entre tecnologias pertinentes à realização deste projeto.

O Capítulo 3 descreve a solução proposta, os critérios que levaram às decisões tomadas na conceção da arquitetura e as tecnologias escolhidas para a implementação.

No Capítulo 4 descreve-se a implemetação dos vários componentes, Collector, bibliotecas de comunicação, Publisher e a aplicação móvel dando ênfase a questões técnicas sobre a implementação.

O Capítulo 5 reporta os resultados e testes efetuados sobre a nova arquitetura em comparação com a anterior.

No Capítulo 6, são expostas as principais conclusões desta tese e possíveis desenvolvimentos futuros.

2 Estado da arte

Este capítulo procura descrever, em parte, a fonte de algumas decisões e práticas adotadas no desenvolvimento deste projeto. Não se pretendeu um estudo exaustivo de nenhum destes tópicos, mas uma perspetiva global que fosse ao encontro dos problemas apresentados. Procurou-se também descrever as tecnologias usadas nos mais diversos cenários, quais as vantagens e desvantagens baseadas em alguns comparativos e casos de estudo.

2.1 Escalabilidade e performance

O termo escalabilidade nas últimas décadas tem assumido um papel importante no desenho e implementação de uma arquitetura de software, embora seja aqui inserido no contexto de serviços *web*, esta temática é transversal a todo o tipo de arquiteturas e sistemas de informação.

Com a contínua expansão de conteúdos na internet, o tráfego e utilização de um determinado site tornou-se uma variável muitas vezes imprevisível e dinâmica. Isto deve-se à exposição pública desses serviços, que fruto do aumento de utilização levanta problemas de performance e no limite a falhas na disponibilidade de serviços. Tornou-se necessário adotar algumas medidas que permitam mitigar este tipo de problemas.

Mas o que é escalabilidade? Um sistema diz-se escalável se for capaz de aumentar a sua utilização e recursos sem que isso provoque perda de performance ou aumente a complexidade de administração [NEUMAN, B. Clifford, 1994].

A escalabilidade de um sistema é medida em três dimensões:

- Tamanho – número de utilizadores ou objetos que fazem parte do sistema
- Questões geográficas – distância entre os nós de um sistema.
- Questões administrativas – o sistema mantém-se gerível mesmo com o aumento do número de organizações.

Em relação ao tamanho, são levantados problemas como: sistemas centralizados que são implementados em apenas um servidor num sistema distribuído; dados centralizados, provocando sobrecarga num único nó; algoritmos centralizados, um servidor apenas que calcula e efetua todas as decisões, ou então servidores que tomam decisões com base em informação de outros nós, caso um falhe o algoritmo não é capaz de continuar e falha também.

As questões geográficas estão relacionadas com o facto de os utilizadores e os recursos estarem distantes uns dos outros. Num sistema distribuído as comunicações são normalmente síncronas, fazendo com que existam dificuldades em ampliar os serviços existentes.

Em relação à gestão administrativa de um sistema, questões relacionadas com a conciliação de politicas de utilização, recursos e segurança são levantados e nem sempre fáceis de resolver.

De acordo com o autor já referido [NEUMAN, B. Clifford, 1994], existem três técnicas de escalabilidade:

- Ocultar a latências na comunicação
- Distribuição
- Replicação

Para esconder a latência na comunicação recomenda-se o uso, sempre que possível, de comunicações assíncronas, para evitar o bloqueio de processos à espera de respostas. As respostas devem ser tratadas como eventos. Usar várias tarefas em paralelo para continuar o processamento e assim ocultar o tempo gasto em comunicações.

Em relação à distribuição, trata-se de decompor um componente em partes mais pequenas, que são distribuídas por todo o sistema. Exemplos desta abordagem é a *web* e servidores de DNS.

A replicação, promove o aumento da disponibilidade dos serviços nos nós de sistema onde mais necessita, procurando balancear a carga entre componentes de uma forma uniforme. Uma forma de replicação é *caching*. Esta abordagem embora muito usada, levanta alguns problemas de consistência de dados. Caso existam várias réplicas de um serviço, e num dos nós é atualizada alguma informação, é importante que as restantes réplicas reflitam a mesma informação. Nalguns casos este tema é tão sensível que esta abordagem é totalmente desaconselhada.

As medidas e técnicas referidas nestes últimos parágrafos foram elaboradas por [NEUMAN, B. Clifford, 1994] há quase 2 décadas atrás, permanecendo atuais nos dias de hoje. No entanto,

com o aparecimento de modelos computacionais como o *cloud computing* foram acrescentados alguns princípios chave que influenciam o desenho de um sistema *web* de grande escala [MATSUDAIRA, Kate, 2012]:

- *Disponibilidade* – a disponibilidade de um serviço é absolutamente crítica à reputação e funcionalidade de um site. Em alguns casos o *downtime* de um serviço pode ter custos elevados. A alta disponibilidade de um sistema distribuído requer considerações cuidadosas em relação à redundância para os componentes principais. Em caso de falha parcial do sistema devem existir mecanismos de rápida recuperação. E caso aconteçam problemas mais graves, os mesmos devem ser apresentados de uma forma "graciosa", como por exemplo com uma página de manutenção.
- *Performance* – o desempenho de um site é um fator importante para a maioria dos sites. A velocidade de um site afeta a satisfação do utilizador bem como o *ranking* atribuído pelos motores de pesquisa, fator que está diretamente relacionado com a visibilidade do serviço. Neste sentido é importante ter um sistema otimizado a latências baixas e respostas rápidas.
- *Fiabilidade* – um sistema deve ser fiável, de modo a que os resultados obtidos sejam consistentes.
- *Escalável* – qualquer sistema distribuído deve ser escalável, como já foi referido aqui, o tamanho é apenas um aspeto de escala que deve ser considerado. Escalabilidade pode referir-se a muitas variáveis diferentes de um sistema, nomeadamente a quantidade de tráfego suportado, capacidade de armazenamento, ou mesmo número de transações podem ser processadas.
- *Gestão* – é importante ter um sistema fácil de gerir. Com já foi referido a gestão está relacionada com a escalabilidade das operações de manutenção e atualizações. Fatores a ter em consideração na gestão são, a facilidade de diagnosticar e compreender os problemas quando eles ocorrem, a facilidade de fazer atualizações ou modificações, e a simplicidade de operações no sistema.
- *Custo* – é um fator importante, está relacionado com custos de hardware e software, mas também devem ser considerados aspetos como a implementação e manutenção do sistema, a quantidade de tempo gasto no desenvolvimento do sistema, e até mesmo a formação necessária.

Cada um destes princípios são critérios importantes para o desenho de um sistema *web* distribuído e escalável, no entanto, alguns critérios podem estar em contradição de tal modo que a realização de um objetivo pode prejudicar outro. Por exemplo, adicionar mais servidores a um sistema é uma questão de escalabilidade que afeta de certo modo questões relacionadas com gestão e custos. Cabe ao arquiteto escolher os princípios mais importantes para o seu sistema.

Uma questão relacionada com a escalabilidade já referida por [NEUMAN, B. Clifford, 1994], é o tamanho/dimensão. Pode existir um elevado número de dados cujo armazenamento não é comportável em apenas um servidor, ou os recursos de computação não sejam suficientes

para as necessidades de computação. Nesta situação existem duas opções de escalabilidade: vertical e horizontal.

- *Escalabilidade vertical*: significa adicionar mais recursos a um servidor. Mais recursos de armazenamento para questões relacionadas com limitações de espaço. No caso de limitações de processamento é possível aumentar CPU ou mesmo memória física. Em cada um dos casos, a escalabilidade vertical não é mais nem menos que aumentar as características do servidor.
- *Escalabilidade horizontal*: significa adicionar mais nós ao sistema. Para questões relacionadas com grandes quantidades de dados, estes podem ser armazenados em vários servidores cada um com um grupo específico de dados. No que diz respeito ao processamento de dados, uma das técnicas mais comuns é dividir um processo em vários serviços e distribuí-los por vários servidores. Estas divisões podem ser distribuídas de forma que cada conjunto lógico possa funcionar separadamente. Para ser possível tirar o máximo partido da escalabilidade horizontal, a mesma deve ser ponderada na fase de desenho da arquitetura de sistema, caso contrário, pode ser complexa a sua implementação.

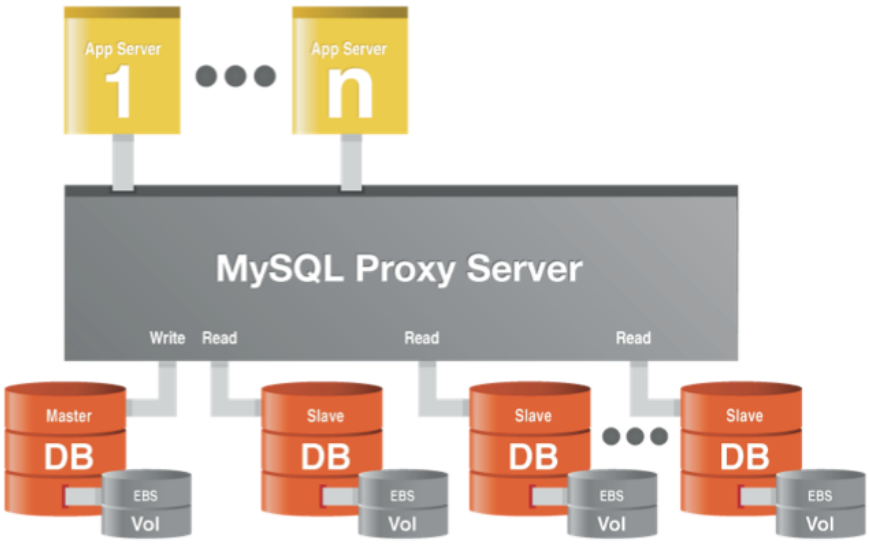


Figura 3 - Escalabilidade horizontal [ADLER, Brian, 2011]

Uma peça fundamental num sistema distribuído e, neste caso, na escalabilidade horizontal é um balanceador de carga (*Load Balance).* A sua funcionalidade é distribuir as tarefas/carga pelos recursos/nós disponíveis. Isto permite que vários nós sejam adicionados ou retirados de uma forma transparente para o utilizador final. Como se pode verificar na figura 4 os vários utilizadores só têm conhecimento de um serviço desconhecendo que existe mais que um nó a funcionar. Esta técnica também permite evitar falhas no sistema, caso um nó falhe isso não vai comprometer por completo o sistema, sendo possível substituir o nó mantendo os restantes a funcionar.

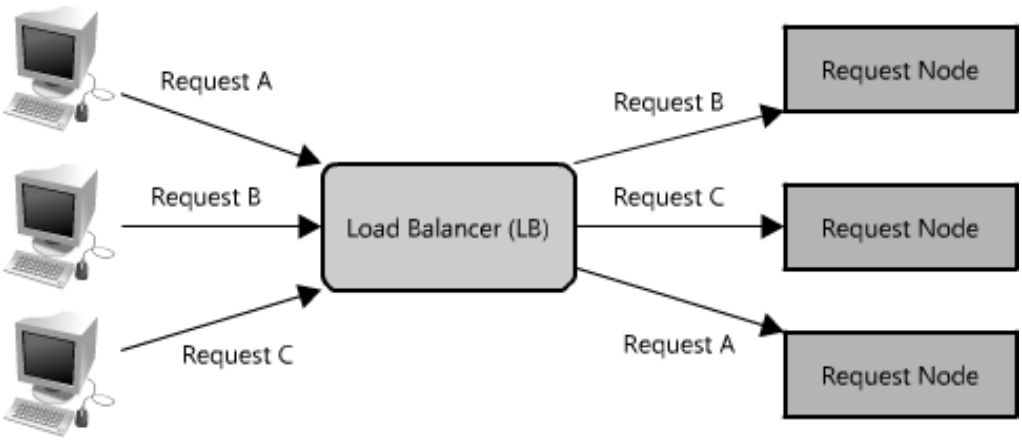


Figura 4 - Load balance [MATSUDAIRA, Kate, 2012]

Um outro mecanismo usado de forma a promover a escalabilidade de uma aplicação na *web* são os *Content Delivery Network* (CDN). Trata-se de sistemas de alto desempenho distribuídos por vários *datacenter's* através internet. O seu objetivo é distribuir conteúdos para o utilizador final, diminuindo assim o tráfego de dados entre o servidor real e o utilizador. A maioria dos servidores de CDN conhecidos funcionam também como servidores de *cache*, caso um determinado conteúdo esteja no CDN e se encontre num período válido o mesmo pedido não necessita de ser solicitado ao servidor real, sendo de imediato disponibilizado ao cliente, caso contrario o pedido é efetuado ao servidor real, entregue ao cliente e armazenado no CDN.

A temática de escalabilidade e performance é extensa e na atualidade é um tema importante em qualquer arquitetura de sistema. Procurou-se na abordagem anterior referir apenas alguns dos tópicos pertinentes a este projeto de Mestrado.

2.2 Message queue

As filas de mensagens têm sido usadas no processamento de dados há vários anos, sobretudo em sistemas distribuidos permitindo troca de mensagens entre componentes. As filas de mensagens fornecem um protocolo de comunicação assíncrona o que siginfica que o produtor e o consumidor de mensagens não necessitam de interagir com o sistema ao mesmo tempo, permitindo total independência um do outro. Por mais complexo que se seja o sistema de mensagens a sua utilização é simples. O produtor coloca as mensagens de acordo com os critérios que pertende dar à mensagem e os consumidores acedem à fila e processam as respetivas mensagem.

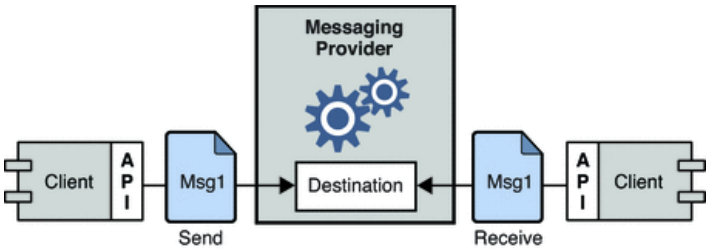


Figura 5 - Message Oriented Middleware [TEQLOG]

Neste projeto de Mestrado, será adotada a técnica de replicação de serviços de forma a garantir uma escalabilidade horizontal do sistema. Neste contexto o autor considerou que o recurso a filas de mensagens seria uma solução óptima para a partilha de informação entre os vários nós. Para este propósito investigaram-se vários serviços de *message queue,* dos quais foram apenas considerados três, fruto da opinião recolhida nas comunidades de desenvolvimento:

- *RabbitMQ* – é um servidor de *message broker* que implementa o protocolo *Advanced Message Queuing Protocol* (AMCP), implementa uma arquitetura de *broker*, o que significa que as mensagens são colocadas numa fila, num nó central antes de serem entregues aos clientes. Esta abordagem torna o RabbitMQ fácil de usar e implementar simplificando questões como roteamento, balanceamento de carga ou filas de mensagens persistentes com apenas algumas linhas de código. No entanto, esta simplicidade torna o servidor menos escalável e "mais lento", porque um nó central acrescenta latência aos envelopes de mensagens.
- *ZeroMQ* – é um sistema de mensagens leve, projetado especialmente para alto desempenho e cenários de baixa latência. O ZeroMQ suporta muitos cenários de troca de mensagens alguns deles avançados e complexos, no entanto, ao contrário do RabbitMQ as implementações de ZeroMQ não são tão simples, é necessário conhecimentos de padrões de mensagens e as questões relacionadas com roteamento e balanceamento de carga não são contemplados nesta tecnologia devendo ser implementados manualmente.
- *ActiveMQ* – é o servidor de *message queue* mais utilizado. Implementa um servidor de *message boker* à semelhança do RabbitMQ e ao mesmo tempo implementa topologias do tipo P2P à semelhança do ZeroMQ. Para além disso oferece, um elevado nivel de suporte com várias tecnologias e *framework* já conhecidas na indústria. No entanto, devído ao elevado número de funcionalidades que oferece a sua simplicidade na implementação acrescenta custos de desempenho.

Para melhor compreender as diferenças entre estas três tecnologias procurou-se comparar o desempenho entre ambos. De acordo com o trabalho realizado por [SALVAN, Muriel, 2013], foram realizados quatro testes a várias tecnologias de *message queue* de forma a apurar o seu desempenho:

- *Cenário A*: *Enqueuing* 20,000 mensagens de 1024 *bytes* cada, *dequeuing* imediatamente a seguir.
- *Cenário B*: *Enqueuing* e *dequeuing* simultaneamente 20,000 mensagens de 1024 *bytes* cada.
- *Cenário C*: *Enqueuing* and *dequeuing* simultaneamente 200,000 mensagens de 32 *bytes* cada.
- *Cenário D*: *Enqueuing* and *dequeuing* simultaneamente 200 mensagens de 32768 *bytes* cada.

Seguem-se os resultados apurados por [SALVAN, Muriel, 2013]:

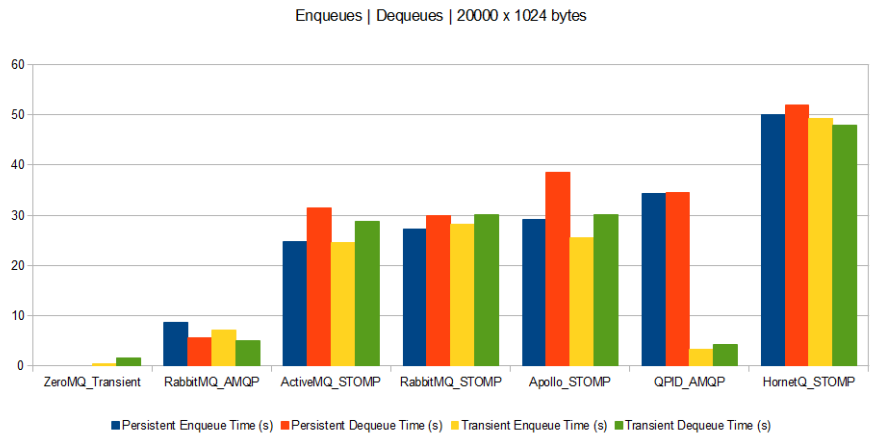


Figura 6 - Message queue benchmark, cenário A [SALVAN, Muriel, 2013]

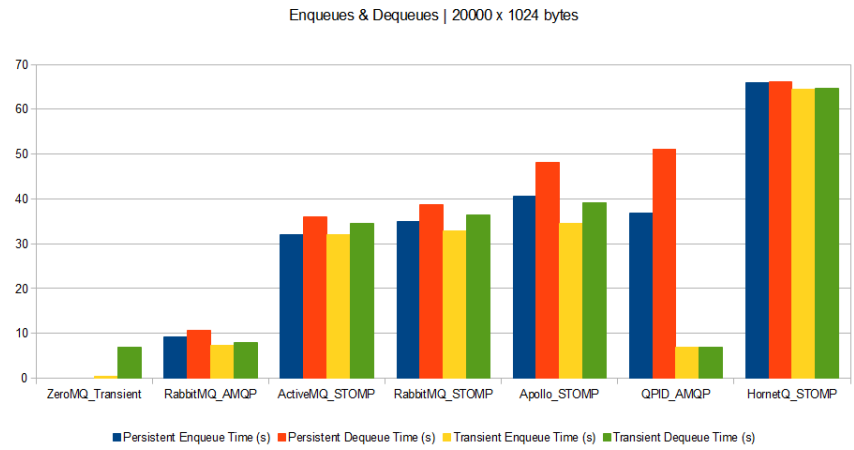


Figura 7 Message queue benchmark, cenário B [SALVAN, Muriel, 2013]

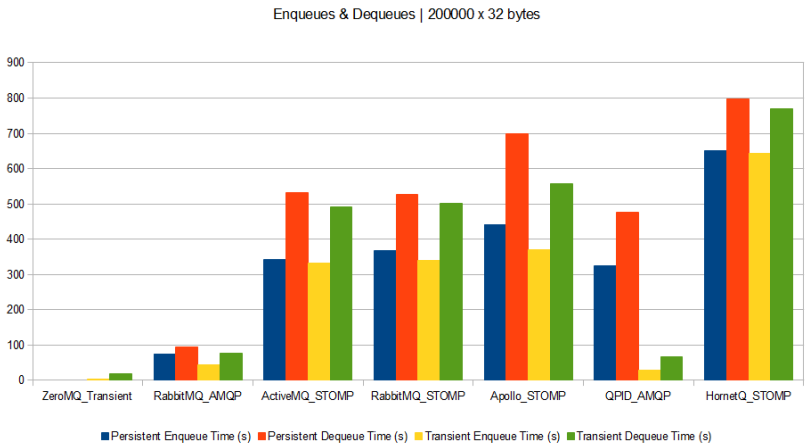


Figura 8 - Message queue benchmark, cenário C [SALVAN, Muriel, 2013]

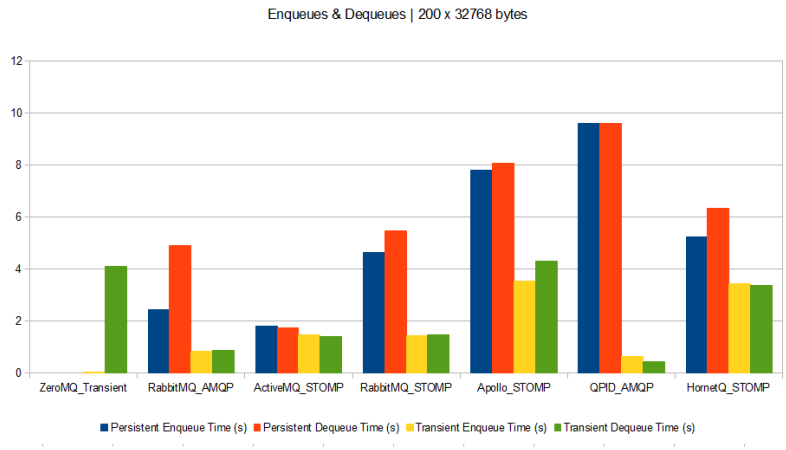


Figura 9 - Message queue benchmark, cenário D [SALVAN, Muriel, 2013]

A primeira impressão que se pode tirar destes resultados é que o ZeroMQ é claramente uma tecnologia com elevado desempenho. Apenas em *dequeuing* de mensagens grandes é que os resultados não foram tão satisfatórios em comparação com as restantes tecnologias. Tornou-se também evidente porque que apenas se consideraram estas três tecnologias embora existam características que aqui não estão a ser consideradas.

2.3 SOA e interoperabilidade entre sistemas

Investigadores e engenheiros de software têm se esforçado ao longo das últimas décadas em simplificar o processo de desenvolvimento de software. Este movimento tem sido impulsionado como forma de acelerar o processo de desenvolvimento e controlo de custos associados ao desenvolvimento e manutenção de software. Nesse sentido alterações nos paradigmas e arquiteturas de software têm vindo a revolucionar a maneira como aplicações são construidas [LIU, Henry H., 2009].

Uma dessas arquiteturas é o SOA (*Service-Oriented Architecture*), uma arquitetura de software cooperativa que promove que as funcionalidades de uma aplicação sejam disponibilizadas na forma de serviços. Cada serviço deve ser capaz de realizar uma determinada tarefa de forma autónoma ou em colaboração com outro serviço, com o objetivo de executar uma tarefas mais complexa. Esta arquitetura é baseada nos princípios de sistemas distribuídos e utiliza um padrão de *request/reply* entre os vários serviços. Estes serviços comunicam entre si através de uma interface, um contrato acessível através de um *webservice* ou outra forma de comunicação entre aplicações. O objetivo é dividir tarefas em serviços e reutilizar serviços existentes.

Segundo a W3C uma arquitetura SOA é tipicamente caracterizada pelas seguintes caracteristicas [Web Services Architecture, 2004]:

- *Visão Lógica*: o serviço é uma abstração lógica da aplicação, abstraindo a sua base de dados, processos de negócio, etc.
- *Orientado a Mensagens*: o serviço é definido formalmente em termos de mensagens trocadas entre *resquest* e *replay* e não as propriedades dos intervenientes. O funcionamento interno desde a implementação, estruturas e processos são deliberadamente abstraídos no SOA.
- *Orientado à descrição*: o serviço é descrito por meta dados, processáveis por máquinas, de forma à exposição pública de um serviço. A semântica de um serviço deve ser documentado , seja direta ou indiretamente, pela sua descrição.
- *Orientado à rede*: serviços tendem a ser orientados para a utilização numa rede, embora não seja um requisito obrigatório.
- *Granularidade*: os serviços tendem a usar um pequeno número de operações com mensagens relativamente grandes e complexas.
- *Neutro de plataformas*: as mensagens são enviadas num formato padronizado e independente da plataforma e entregues através de interfaces.

Como se pode verificar esta aquitetura promove um conjunto de características usadas em sistemas distribuídos, permitindo também a interoperabilidade entre sistemas heterogéneos. A arquitetura em si não define o modo como os serviços são implementados, apenas define que devem comunicar através de um meio comum. Assim é possível a interoperabilidade entre sistemas, plataformas e linguagens distintas.

Num sistema distribuído existe um conjunto de desafios ao nível da arquitetura:

- Problema da latência e fiabilidade do canal de transporte
- A falta de memória partilhada entre os intervenientes (quem chama o serviço e quem responde).
- Problemas relacionados com falhas parciais.
- Concorrência entre pedidos
- A incompatibilidade de dados entre os intervenientes.

Estes desafios aplicam-se a todos os sistemas distribuídos independentemente do meio de comunicação que usam numa arquitetura SOA. No contexto deste projeto, entendeu-se apenas estudar os *webservices* como meio de comunicação. O autor está consiente que o uso de *webService* por si só não transforma um sistema distribuído numa arquitetura SOA nem necessariamente a melhor escolha para a sua implementação. No entanto, tendo em conta as características deste projeto, relacionadas com um sistema distribuído com base na *web* e com elevada heterogeneidade de plataformas, o autor considerou que o uso de tecnologias *webservices* seriam apropriadas e os benefícios do seu uso compensariam os problemas de desempenho e implementação que pudessem introduzir.

Os *webservices* em termos gerais são um método de comunicação entre duas aplicações ou dispositivos através da internet (WWW). Os *webservices* são conhecidos por dois tipos: *Simple Simple Object Access Protocol* (SOAP) e *Representational State Transfer* (REST).

- *SOAP* - define um protocolo de comunicação padrão (conjunto de regras) de troca de mensagens baseado em XML. É possível usar SOAP sobre diversos protocolos: HTTP, JMS e SMTP, embora seja mais utilizado sobre o protocolo HTTP por tornar o processo comunicação mais simples evitando questões relacionadas com *proxies* e *firewall.* O SOAP acrescenta um elevado nível de descrição de objetos e serviços em formato XML permitindo assim interoperabilidade entre plataformas .
- *REST* – é um conjunto de princípios que define como HTTP e os URI devem ser usados na *web*. O REST não adiciona nenhuma camada à já existente pelo protocolo. Os serviços são disponibilizados nos URIs e a utilização dos métodos GET, POST, PUT, DELETE e HEAD do protocolo HTTP define as ações que esses serviços vão efetuar.

Existem vários critérios para a escolha do tipo de *webservice* que deve ser implementados. Ainda nos dias de hoje o SOAP é uma referência e usado em toda a indústria como um excelente meio de interoperabilidade, devido à sua elevada descrição de serviços e objetos. Embora, por um lado, a descrição detalhada dos serviços seja algo positivo, por outro lado aumenta os dados enviados pelo canal de comunicação aumentando assim o tempo de comunicação e tráfego utilizado. O REST em parte evita este problema, como não existe uma descrição formal do serviço e o tipo de dados que é trocado não obriga a um padrão tipo o XML, permite que apenas seja enviada a informação estritamente necessária. Esta abordagem é vantajosa quando se pretende oferecer serviços a sistemas móveis.

2.4 HTTP e comunicações em "tempo real"

Numa aplicação *web* a comunicação entre o cliente e o servidor é sempre iniciada por parte do cliente. Este processo de *pull* por parte do cliente acontece sempre que necessita de nova informação.

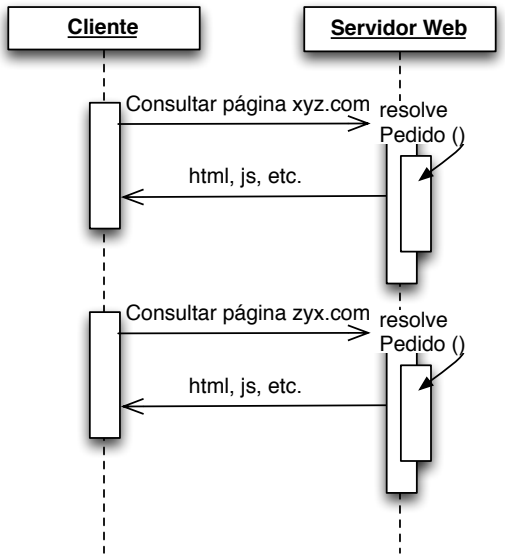


Figura 10 - Diagrama de sequências de pedidos Web

Não é possível o servidor enviar respostas para os seus clientes sem existir um pedido por parte deste.

Existem vários casos onde este processo levanta algumas limitações a esta arquitetura, como por exemplo, quando um cliente consulta uma determinada informação e passado alguns segundos essa informação fica desatualizada. Uma das técnicas usadas para ultrapassar este problema passa por efetuar periodicamente consultas ao servidor através de mais pedidos *pull*. Embora esta solução fosse razoável e ainda usada nos dias de hoje, são vários os casos onde esta abordagem é pouco satisfatória. Exemplos como a bolsa, jogos on-line, resultados financeiros e serviços noticiosos são alguns dos casos onde a abordagem referida pode trazer alguns problemas e limitações.

Embora existissem vários protocolos de comunicação tendo de base a internet, como o Instant messaging, Internet Telephony & VoIP entre outros, tratava-se de serviços proprietários que necessitavam de *software* externo e funcionavam num contexto fechado não sendo possível transportá-los para a *web*.

Surgiram as tecnologias RIA como é o caso: Adobe Flash, Microsoft Silverlight e as applet Java, que em grande parte vieram enriquecer os conteúdo da *web*, permitindo também estabelecer comunicações instantâneas entre o servidor e o *browser*, mas mais uma vez necessitavam de software externo.

Com o evoluir dos anos a *web* deixou de ser apenas um serviço de hipertexto tornando-se em algo que é difícil definir os seus limites. Nestes últimos anos temos assistido a um conjunto de serviços que tendem em passar para o *browser* através do modelo de "*cloud computing*", que há poucos anos atrás apenas seriam possíveis através da instalação de novo *software*. Isto é fruto da evolução tecnológica dos *browser's* mais recentes e novas especificações nas mais diversas áreas.

É neste sentido que se procurou observar as tecnologias e técnicas mais utilizadas com o objetivo de implementar comunicações instantâneas na *web*.

2.4.1 WebRTC

O WebRTC trata-se de um padrão de comunicação desenvolvido pela W3C em cooperação com o padrão RTCWeb desenvolvido pela IETF. Na camada inferior do protocolo são usadas funções RTCWeb; WebRTC permite incorporar essas funcionalidades em aplicações e websites.

O padrão WebRTC soluciona vários problemas comuns, como por exemplo, incompatibilidade entre comunicações em tempo real e possibilidade de serem efetuadas comunicações *browser-to-browser* independentemente da plataforma sem qualquer aplicação adicional. Este padrão permite a comunicação entre *browser's* de vídeo, áudio e dados [VOXEO].

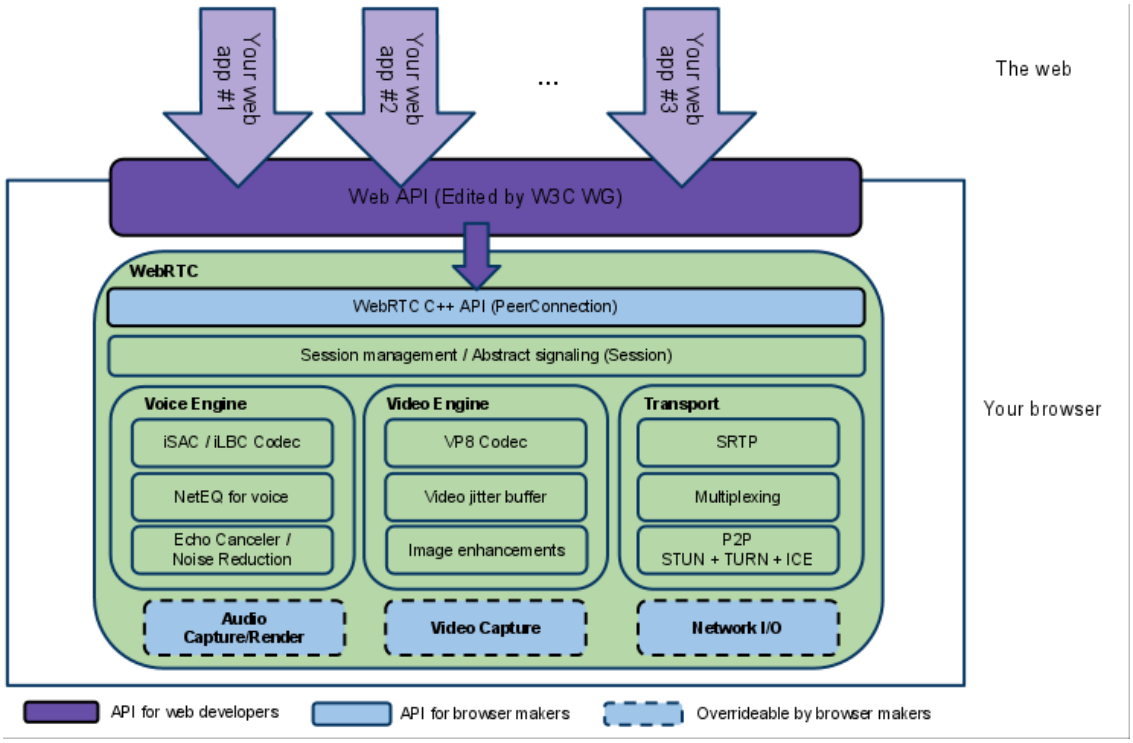


Figura 11 - Arquitetura WebRTC [WEBRTC, 2013]

Com a web API disponibilizada no WebRTC é possível desenvolver um conjunto de aplicações que permitam a comunicação em tempo real entre *browser's* de uma forma transparente sem a necessidade de qualquer *software* externo.

No entanto, até à presente data esta tecnologia está presente em apenas dois *browser´s* (Google Chrome e Mozilla Firefox), existem ferramentas que permitem manter a compatibilidade com os outros *browser's* como é o caso de webrtc4all[1], mas nota-se que existe ainda um longo caminho a percorrer até esta tecnologia se tornar padrão entre todos eles.

2.4.2 Comet e Reverse Ajax

Comet é um modelo de aplicações web onde os pedidos enviados ao servidor são mantidos ativos por um período longo de tempo tornando as ligações HTTP persistentes. Quando é enviada uma resposta do servidor, o cliente efetua novo pedido de forma a manter uma nova ligação ativa. Com este modelo, os servidores *web* podem enviar dados aos seus clientes sem existir um pedido explícito [CARBOU, Mathieu, 2011]. Esta abordagem também é conhecida por outros nomes como *Ajax Pus*h, *Reverse Ajax*, *Two-way-web*, *HTTP Streaming* e *HTTP server push*.

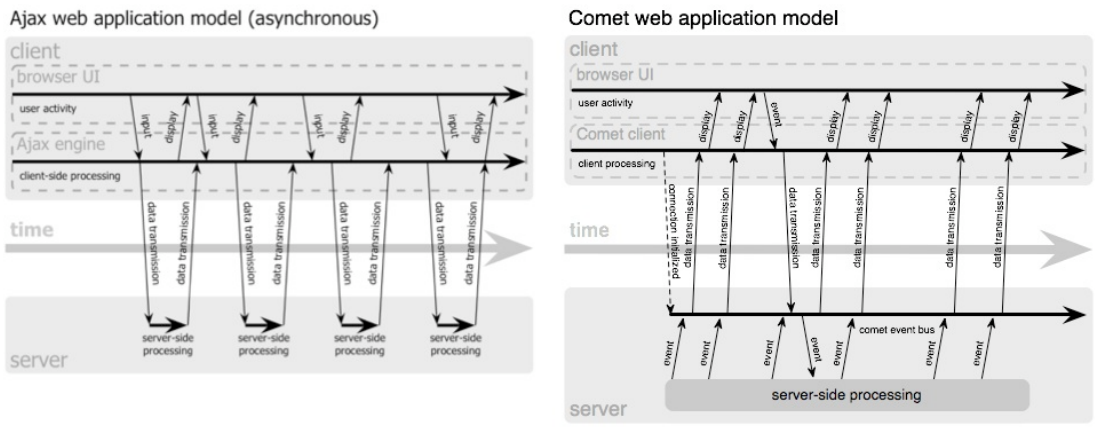


Figura 12 - *Ajax Web Application model* vs *Comet Web Application Model* [Evolution of COMET, 2008]

A grande vantagem do Comet é que cada cliente tem sempre uma ligação aberta com o servidor, permitindo assim que ele consiga enviar mensagens *push* de uma forma imediata. Para este modelo ser possível é necessário algumas funcionalidades no servidor *web* e como é óbvio código específico no lado do cliente.

[1] https://code.google.com/p/webrtc4all/

São várias as aplicações que usam esta abordagem, só que existe um problema, o protocolo HTTP não é adequado a aplicações com baixa latência.

2.4.3 WebSocket's

O HTML5, a mais recente versão da linguagem que permite estruturar e apresentar conteúdo na *web*, trouxe nesta versão importantes alterações ao papel que o HTML poderia ter no mundo *web*. Resultado de vários anos de discussão e especificação deste *standard,* que envolveu vários fabricantes, está a tornar esta especificação amplamente aceite por todos os fabricantes de *browser's* e aclamado pela comunidade de programadores [PETER , Lubbers et al., 2010; VOXEO]. Mas porquê todo este entusiasmo? A resposta é simples, o HTML5 veio resolver um conjunto de problemas práticos, como aquele que está a ser descrito neste capítulo. Entre várias novas funcionalidades que este padrão acrescenta, surgem os WebSocket's.

WebSocket é uma tecnologia que permite a comunicação bidirecional, full-duplex, entre o browser e o servidor *web* com suporte de HTML5. A especificação de WebSocket define uma API que permite estabelecer uma ligação através de um único "*socket*" TCP entre o browser e o servidor, trata-se de uma ligação persistente entre os intervenientes onde ambos podem enviar dados a qualquer momento. Embora esta tecnologia tenha sido desenvolvida para ser implementada em browsers e servidores *web*, a mesma pode ser usada por qualquer cliente ou servidor. É um protocolo independente baseado em TCP cuja única relação com o protocolo HTTP é que o seu *handshake* é interpretado por servidores HTTP com um pedido semelhante ao que se segue:

Do cliente para o server:	Do servidor para o cliente:
GET /demo HTTP/1.1 Host: example.com Connection: Upgrade Sec-WebSocket-Key2: 12998 5 Y3 1 .P00 Sec-WebSocket-Protocol: sample Upgrade: WebSocket Sec-WebSocket-Key1: 4@1 46546xW%0l 1 5 Origin: http://example.com [8-byte security key] From server to client: HTTP/1.1 101 WebSocket Protocol Handshake Upgrade: WebSocket Connection: Upgrade WebSocket-Origin: http://example.com WebSocket-Location: ws://example.com/demo WebSocket-Protocol: sample	HTTP/1.1 101 WebSocket Protocol Handshake Upgrade: WebSocket Connection: Upgrade WebSocket-Origin: http://example.com WebSocket-Location: ws://example.com/demo WebSocket-Protocol: sample

Tabela 1 - *Handshake* de pedidos WebSocket

Os WebSocket são uma tecnologia recente, razão pela qual ainda se encontra pouco disseminada pela indústria. Os servidores web nem todos disponibilizam suporte e apenas recentemente é que a maioria dos browsers suportam esta tecnologia.

	Global
Suporte :	68,47%
Suporte parcial:	2,87%
Total:	71,34%

Tabela 2 – Suporte de browsers a WebSocket [Can I use, support tables for HTML5, CSS3, 2013]

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	IE Mobile	
	8		28			5		4			
	9	23	29	5.1		6		4.1	7		
Versão corrente	10	24	30	6	17	7	5 7	4 2	10	10	
Futuras versões	11	25	31	7							

Tabela 3 - Browsers compativeis com WebSocket's [Can I use, support tables for HTML5, CSS3, 2013]

Embora a percentagem 71% de suporte seja desanimadora, atualmente os browsers mais utilizados já oferecem suporte a esta tecnologia motivando a sua utilização.

2.4.4 Socket.IO

Socket.IO é uma biblioteca JavaScript que oferece uma API única, semelhante a WebSocket, que permite estabelecer ligações a servidores remotos de forma a ser possível enviar e receber mensagens num modo assíncrono. Disponibiliza uma API comum que suporta vários canais de transporte, tais como, WebSocket, Flash Sockets, *long polling, streaming, Iframes* e *JSONP polling*. Socket.IO deteta a compatibilidade com o *browser* e escolhe qual o melhor canal de comunicação disponível. Esta biblioteca é compatível com um elevado número de *browser's* incluindo os mais antigos como o Internet Explorer 5.5 [CARBOU, Mathieu, 2011; LIU, Henry H., 2009].

O Socket.IO é composto por duas partes distintas, uma biblioteca JavaScript para o lado do cliente (*Browser's*) e uma biblioteca para o servidor Node.JS (plataforma de JavaScript que permite criar aplicações de uma forma simples e rápida). É de referir que embora a biblioteca disponibilizada pelo Socket.IO para servidor seja em JavaScript (Node.JS), existem várias

implementações de terceiros que permitem utilizar o cliente desta biblioteca em outro tipo de linguagens como Erlang, Java, Lua, Object-C, C, C++, Python, Flash, .NET entre muitas outras.

2.5 Desenvolvimento em plataformas móveis

Temos vivido nestes últimos anos um crescimento no desenvolvimento de aplicações móveis. Todas as plataformas móveis oferecem pelo menos uma forma de se criar aplicações. Quando comparadas com os websites móveis ou os widgets, estas aplicações oferecem uma maior integração com o sistema operativo que se reflete, quase sempre, num melhor aproveitamento das suas funcionalidades.

Atualmente existem várias plataformas móveis disponíveis no mercado e são, na sua generalidade, incompatíveis umas com as outras quanto à criação de aplicações. Ter uma presença alargada neste mercado representa, normalmente, a criação de várias versões da mesma aplicação de forma a conseguir-se aproveitar as capacidades de cada uma das plataformas.

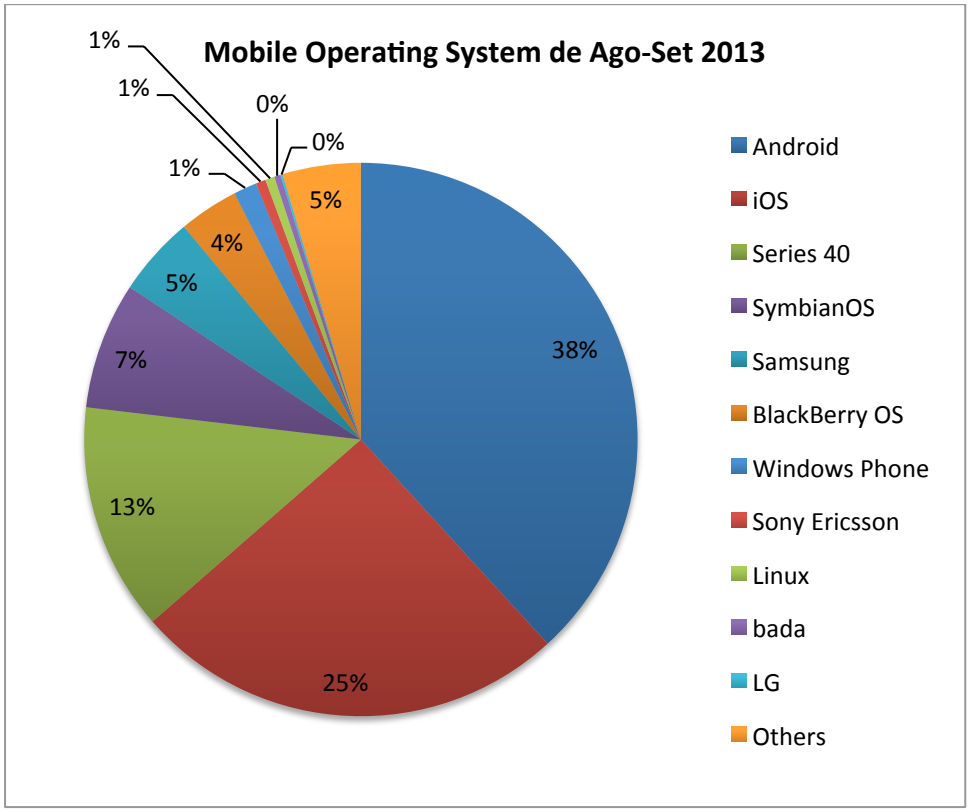


Figura 13 - Uso de plataformas móveis (Fonte statcounter.com)

Como se pode verificar na figura 6, atualmente 63% do mercado é divido pelas duas plataformas que mais têm crescido estes últimos anos, Android e iOS. O desenvolvimento apenas para estas duas plataformas implica que 37% deste mercado não será abrangido. Mais ainda, estas percentagens têm tendência a sofrer algumas alterações em períodos curtos de

tempo, como foi o caso da plataforma Symbian e BlackBerry OS que já tiveram outra expressividade.

É evidente que o desenvolvimento de aplicações nativas para cada uma das plataformas seria o ideal, de modo a tirar o máximo partido dos recursos que cada uma das plataformas oferecesse. Mas manter várias versões da mesma aplicação implica um esforço adicional ao nível de recursos técnicos. Para cada uma das plataformas seria necessário conhecimentos específicos, algo que não é possível em alguns projetos, por questões de tempo ou não terem capacidade de ter equipas de desenvolvimento específicas para cada uma das plataformas.

2.5.1 Multiple devices application framework

Para resolver algumas das questões relacionadas com a compatibilidade entre diversas plataformas, surgem algumas *frameworks* de desenvolvimento para plataformas móveis que permitem o desenvolvimento de aplicações compatíveis com diversas plataformas. As *multiple devices application framework* são *framework's* de software desenhadas para suportar o desenvolvimento de aplicações móveis numa determinada linguagem. Disponibilizam através da *framework*, funcionalidades nativas de cada plataforma, como por exemplo, câmera, acelerómetro, geolocalização, etc, funcionalidades que são comuns a todas as plataformas. Estas *framework's* funcionam como uma camada de *middleware* entre o código e uma plataforma móvel. As vantagens desta abordagem são evidentes, como o rápido desenvolvimento, código comum para várias plataformas, no entanto, as desvantagens também são relevantes, sendo o desempenho e as funcionalidades restritas à *framework*. É importante ter presente quais os objetivos da aplicação móvel, há casos que se torna evidente que a adoção de uma tecnologia deste género não é a mais adequada, sobretudo quando se pretende tirar o máximo partido do hardware ou mesmo das funcionalidades que cada sistema operativo oferece, como por exemplo a indústria de jogos. Mas existem vários casos práticos onde a adoção destas *framework* pode trazer benefícios .

É neste contexto que foram analisadas alguns das *framework's* que mais se destacam nesta abordagem.

Foram utilizados os seguintes critérios de comparação: simplicidade no desenvolvimento, o maior número de dispositivos suportado, compatibilidade com comunicações em tempo real, uma elevada compatibilidade com as plataformas nativas e licença *open-source*.

	Appcelerator Titanium	codenameone	Kivy	MoSync	Apache Cordova
Linguagem	HTML, JavaScript	Java	Python	C/C++, JavaScript, HTML e CSS	HTML, JavaScript e CSS
iOS	Sim	Sim	Sim	Sim	Sim
Android	Sim	Sim	Sim	Sim	Sim
Windows Phone	-	-	-	Sim	Sim
Blackberry	-	Sim	-	Sim	Sim
Symbian	-	-	-	Sim	Sim
WebOS	-	-	-	-	Sim
Bada	-	-	-	Sim	Sim
Open-source	Sim	Não	Sim	Sim	Sim

Tabela 4 - *Framework's* multiple devices

As *framework's* Titanium, MoSync e Cordova seguem uma abordagem semelhante em termos de implementação, usam a liguagem de HTML5 e Javascript como base, permitindo assim um simples e rápido desenvolvimento, a compatibilidade com as funcionalidade nativas são semelhante entre ambas, destacando-se o Apache Cordova pelo suporte a eventos do acelerómetro e mapas. O suporte de HTML5 nestas *framework's* é limitado a algumas funcionalidades. Para o que se pretende neste projeto, a API de websocket é a funcionalidade que mais interessa nesta tecnologia, mas apenas é suportada em algumas plataformas. Existem algumas técnicas que permitem contornar esta questão, sendo certo que num futuro breve, o suporte a esta tecnologia será incluido em ambas as plataformas.

O Kivy é uma plataforma Python que se destaca pelo desempenho e em comparação com as restantes *framework's*, permite o desenvolvimento de jogos e aplicações gráficas com elevados níveis de performance num ambiente de multi-platafoma. Objetos nativos das plataformas são escassos, embora disponibilize ferramentas para a sua concepção. Apenas suporta as plataformas iOS e Android.

A codenameone, uma *framework* Java, disponibiliza um conjunto interessante de funcionalidades em comparação com as restantes, no entanto a licença de utilização é restrita e não é *open-source*. À semelhança do Kivy oferece suporte limitado de equipamentos, suportando apenas iOS, Android e blackberry.

As conclusões aqui retiradas foram baseadas na documentação que as *framework's* disponibilizam. Ambas anunciam um conjunto de funcionalidades nos seus *roadmap,* no entanto, apenas foram tidas em consideração as funcionalidades que já se encontram estáveis.

3 Solução Proposta

O problema aqui colocado, como já referido no capítulo 1, apresenta alguns problemas de arquitetura, limitando com isso a sua escalabilidade, segurança e latência na entrega da informação. Além disso, a plataforma existente não se encontrava preparada a fornecer serviços a terceiros, como por exemplo, as plataformas móveis.

Após o estudo das várias temáticas abordadas no capítulo 2, tornou-se evidente que seria necessário adaptar e criar novos componentes que substituíssem os existentes. Deste modo verificou-se que existem três componentes distintos, distribuídos e que funcionam isoladamente.

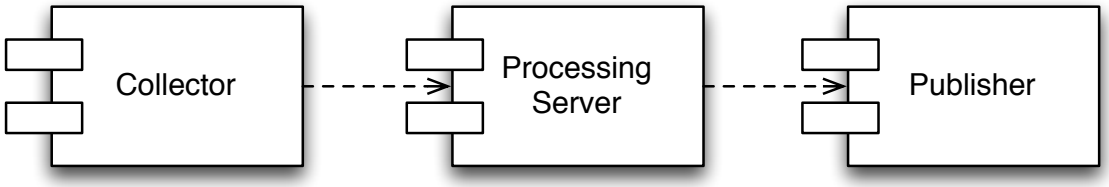


Figura 14 – Componentes

Estes componentes já existiam na antiga abordagem, fruto da evolução que o sistema foi tendo ao longo dos anos. O "Collector" existente consistia num conjunto de ferramentas e utilitários que serviam para registar diretamente numa base de dados os valores recolhidos. Praticamente existia um utilitário para cada tipo de terminal. A proposta apresentada passou por adotar a mesma distribuição de componentes, mas baseada em serviços.

A implementação existente do "Collector" e do "Publisher" tinha um conjunto de limitações técnicas que levaram o autor a decidir a sua substituição. Ambos os componentes estavam implementados em PHP sem nenhuma *framework* de suporte e a sua adaptação a uma nova arquitetura implicaria um desenvolvimento quase de raiz de toda a solução. No que diz respeito ao componente "Processing Server", o elemento central da solução, foi proposta a adaptação dos módulos que comunicam com o "Collector" e o "Publisher" mantendo o restante comportamento esperado para este componente.

3.1 Nova Arquitetura

Um dos objetivos que se pretende neste projeto de Mestrado é transformar os vários serviços dispersos numa arquitetura SOA, permitindo assim a exposição destes serviços a outras aplicações e resolver alguns problemas de segurança como é o caso das bases de dados expostas.

Tendo em conta que os componentes apresentados podem estar distantes geograficamente, houve a necessidade de projetar duas API, uma para o componente "Collector" e outra para o "Publisher", de forma a ambas poderem continuar a trabalhar de uma forma isolada e autonomamente.

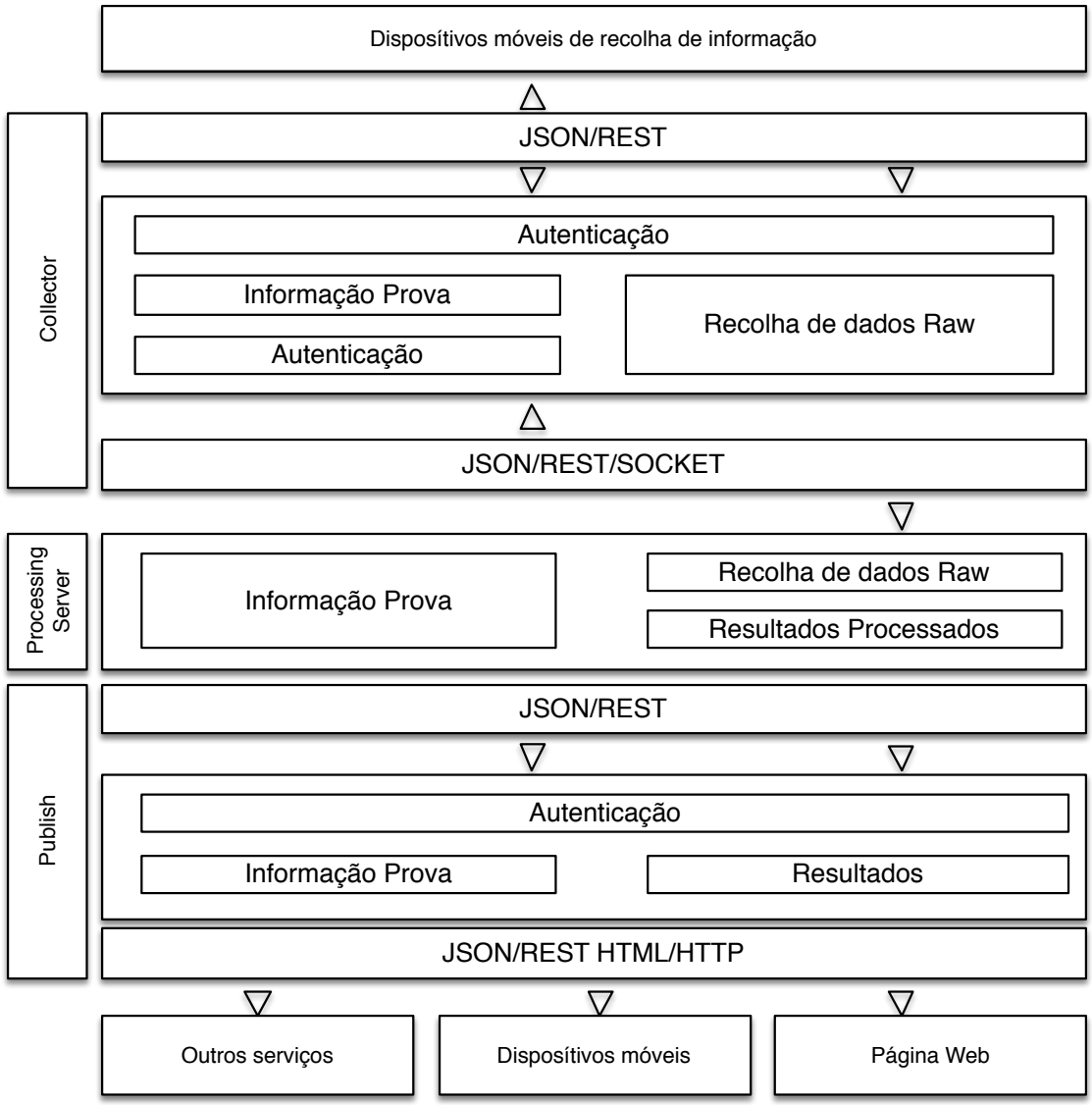


Figura 15 - Nova arquitetura

A API Collector disponibiliza os recursos necessários para recolher informação ainda por tratar. Através de um *webservice* REST os terminais obtêm dados sobre as provas e enviam a informação recolhida nesses terminais. Como se trata de informação sensível, todos os terminais necessitam de efetuar uma autenticação.

Um problema existente na anterior arquitetura era que o servidor de processamento tinha periodicamente de consultar a base de dados de recolha para verificar se existiam novos dados. Neste nova abordagem, existe um *socket* entre o Collector e o Processing Server, permitindo assim que a informação quando chegar ao Collector é imediatamente disponibilizado no servidor de processamento.

O Processing Server através do *webservice* do Collector disponibiliza informação necessária à recolha de dados. Essa informação apenas pode ser enviada após autenticação do servidor com a API Collector.

A API Publisher à semelhança do que acontece com a API Collector disponibiliza um *webservice* REST com uma interface que permite inserir dados e resultados de uma prova mediante autenticação. Disponibiliza também na mesma interface alguns métodos sem autenticação que permitem consultar informação passível de ser consultada.

O serviço de Publisher é mais que uma API, fornece também uma interface web onde apresenta os dados inseridos através da API e estabelece um WebSocket entre o browser e o servidor de Publisher quando as provas se encontram a decorrer de forma a ser possível obter os dados imediatamente no *browser* do utilizador sem qualquer interação ou atualização do utilizador.

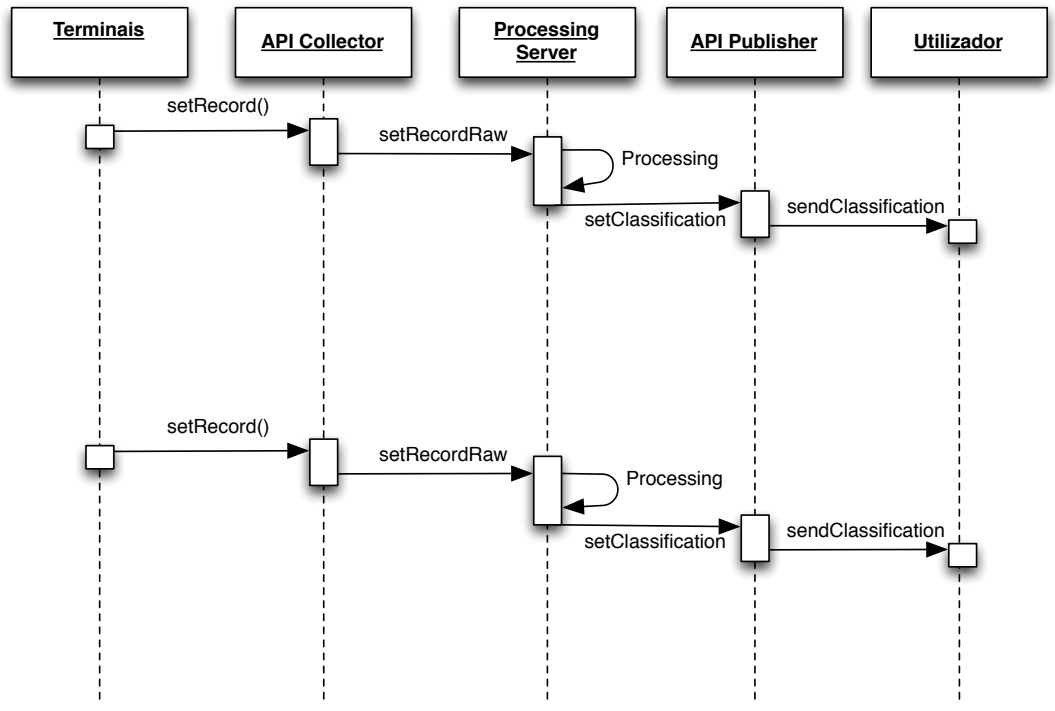


Figura 16 - Diagrama de sequência da nova arquitetura

3.2 Tecnologias

A evolução *web* é uma realidade e com ela surgiu um leque enorme de tecnologias, padrões e abordagens para a sua construção. Trouxe com isto, um problema aos arquitetos de sistemas, "quais as tecnologias a usar para um determinado problema?". A resposta nem sempre é simples, há vários fatores que devem ser considerados, mas antes de partirmos para uma decisão tecnológica devemos ter um objetivo claro do que se pretende.

3.2.1 Linguagens e framework's de programação

Desde o início deste projeto que houve a liberdade de escolher a tecnologia que fosse mais adequada aos objetivos pretendidos. A linguagem de programação foi provavelmente o elemento tecnológico que levou mais tempo a decidir.

Numa primeira fase foi tido em atenção se existia alguma restrição de compatibilidade com os sistemas existentes.

O Collector existente como já referido, seria totalmente reescrito oferecendo apenas serviços com o exterior, não demonstrando qualquer restrição em relação à linguagem de programação a usar.

O Processing Server, tratava-se apenas de uma adaptação dos módulos existentes. Como toda a plataforma é escrita na plataforma .net da Microsoft, a escolha tecnológica neste componente foi simples, manteve-se a linguagem adaptando apenas o código existente.

O Publisher foi o componente onde surgiram dúvidas sobre manter a linguagem existente ou substituir caso necessário, porque como já existia uma aplicação *web*, ponderou-se em se adaptar à nova arquitetura. Depois de uma análise ao código existente e ponderando os objetivos propostos chegou-se à conclusão que a parte do código que poderia ser mantida seria mínima. Deste modo não se encontrou nenhum problema em mudar de linguagem.

Surgiu então a oportunidade de escolher uma linguagem que fosse ao encontro dos objetivos propostos. Para isso foram definidos alguns critérios de forma a ajudar a escolha:

- Otimizada para ambientes Web
- Escalável – embora parte deste ponto seja da responsabilidade do código escrito
- Rápida de aprender e desenvolver
- *Framework* disponíveis
- Simples de manter
- Custos financeiros – licenciamento de software, servidores, etc
- Multiplataforma

Existem várias linguagens que poderiam ser avaliadas perante estes critérios, mas tendo em conta o esforço que isso teria apenas foram consideras algumas relevantes.

- *.net* – É otimizada para *web*, escalável, rápida para desenvolver, aprender e simples de manter. No entanto acresce de alguns custos financeiros de licenciamento e o suporte oficial é apenas para servidores Windows.
- *Java* – Existem várias *framework* que transforma esta linguagem poderosa em aplicações *web*, o suporte é grande, rápida em desenvolvimento, simples de manter, multi-plataforma. A parte negativa que levou o autor a não escolher esta linguagem são os custos de infraestruturas para ter um servidor operacional, comparando com outras plataformas.
- *PHP* – A solução existente é escrita em PHP, embora ofereça todos os critérios propostos, à medida que o projeto foi crescendo tornou-se difícil de manter o código existente, rapidamente fica desorganizado e confuso.
- *Python* – Com a *framework* certa é possível ter uma plataforma poderosa para aplicações *web*. Robusta, escalável, multiplataforma, com uma sintaxe simples e elegante e fácil de manter.

A escolha sobre a linguagem a usar nos componentes Collector e Publisher foi Python. Embora o autor deste projeto não tivesse qualquer conhecimento sobre Python, assumiu esse risco por estar convicto que seria uma plataforma ótima à implementação de *webservices* REST e WebSocket's em comparação com algumas *framework's* de PHP analisadas.

3.2.2 Tornado + Nginx

Tornado é um servidor *web* não bloqueante e uma *framework* para aplicações *web* em Python. Através das ligações não bloqueantes à rede, Tornado é capaz de escalar milhares de ligações em simultâneo, sendo possível criar várias instâncias do mesmo servidor, na mesma máquina, mantendo as restantes em funcionamento, de forma a tirar o melhor partido dos recursos de *hardware*. Trata-se de uma framework simples, que para além das suas funcionalidades é uma ótima solução para aplicações de *long polling* e *websocket's* como a que pretendemos implementar.

O Nginx é um servidor *web opensource* criado com o objetivo de ser altamente concorrente, elevados níveis de performance e uso reduzido de memória. Para além disso, é um *reverse proxy server* para vários protocolos como HTTP e HTTPS. Funciona também como *load balance* e HTTP *cache*. A escolha deste servidor *web* recaiu sobretudo na funcionalidade de *reverse proxy* que oferece uma elevada compatibilidade com o *web server* Tornado.

3.2.3 SQLAlchemy + MySql

Existe a necessidade de armazenar os dados de uma forma persistente. Com o aparecimento dos ORM é possível fazê-lo mantendo um modelo orientado a objetos, mapeando as tabelas em objetos, colunas em atributos e oferecendo um conjunto de funcionalidades que permitem tirar partido dos vários DBMS de uma forma transparente, encapsulando as ligações, *queries* e a lógica relacional de uma base de dados. Existem algumas desvantagens na

utilização destas *framework* de base de dados, a mais evidente é, acrescentar mais uma camada de software podendo afetar assim a performance do sistema. Por outro lado permite tornar o desenvolvimento mais transparente e rápido.

Quando este projeto foi planeado, pretendeu-se também que o sistema tivesse uma arquitetura que fosse fácil e rápida de manter em futuros desenvolvimentos.

O autor deste projeto entendeu que a adoção de um ORM traria vantagens futuras no desenvolvimento da plataforma. Nesse sentido foram avaliadas várias *frameworks* em Python com estas características e rapidamente se chegou ao SQLAlchemy.

O SQLAlchemy disponibiliza um conjunto completo de padrões de persistência de dados conhecidos e usados no universo empresarial. Foi projetado para ser eficiente e com elevado nível de performance na linguagem Python [SQLALCHEMY, 2013].

O MySql é um DBMS *open source* com uma grande comunidade de utilizadores onde tem mostrado grandes níveis de eficiência e performance.

3.2.4 Socket e ZeroMQ

A solução que se propõe para este projeto de Mestrado prevê que exista um elevado número de troca de mensagens entre os vários componentes do sistema. Como se pretende dotar estes componentes de um elevado nível de escalabilidade, existiu a necessidade de implementar mecanismos que permitam a troca de mensagens entre as mais diversas instâncias do mesmo componente. Embora se pretenda implementar duas API que permitam receber mensagens, esta funcionalidade foi desenhada apenas para interoperabilidade entre serviços externos, que não necessitam de conhecer detalhes de como cada componente funciona internamente. No entanto existem casos onde há necessidade de seguir uma outra abordagem, é o caso de: quando as mensagens são rececionadas e necessitam de ser enviadas para o Processing Server; quando os resultados são rececionados pelo Publisher e necessitam de ser difundidos pelos WebSockets abertos; quando o sistema escala e o número de webserver's aumenta e os mesmos necessitam de comunicar entre si.

A opção que o autor adotou para ultrapassar estas questões foi a implementação de um mecanismo de *message queue*, que permitisse que cada componente pudesse comunicar internamente sem afetar a sua performance, permitindo também a sua escalabilidade sem prejudicar o seu funcionamento.

Despois do estudo efetuado no capítulo 2.1 sobre *message queue* e as tecnologias ActiveMQ, RabbitMQ e ZeroMQ o autor entendeu que a solução mais adequada para esta projeto seria o ZeroMQ.

O ZeroMQ é uma biblioteca de *sockets* de alto desempenho que permite o envio de mensagens num modo assíncrono, utilizado na indústria em aplicações distribuídas, concorrentes e escaláveis. Disponibiliza também uma fila de mensagens, mas ao contrario do

ActiveMQ ou RabbitMQ, o ZeroMQ pode ser executado sem um servidor de mensagens dedicado. Permite efetuar ligações P2P diminuindo assim a latência de comunicação, ao contrário dos outros serviços que precisam de uma ligação ao servidor para depositar as mensagens e uma outra ligação para recolher mensagens, aumentando assim *overhead* de comunicação.

A biblioteca ZeroMQ tem um elevado nível de parametrização, permitindo implementar vários *Messaging Patterns,* desde o simples *Request-reply, Pub-sub* ao *Dealer and Router,* mantendo em todos eles a mesma simplicidade e transparência na implementação.

O autor deste projeto tem consciência que serviços como o ActiveMQ ou o RabbitMQ são ótimas soluções, inclusive mais simples de implementar do que o ZeroMQ, mas tendo como principais objetivos a escalabilidade e performance da solução, esta opção foi tomada como a mais adequada para este projeto em particular.

3.2.5 NoSQL redis.io

"NoSQL" é um termo genérico que classifica base de dados não relacionais, tipicamente distribuídas e horizontalmente escaláveis. Este tipo de bases de dados são criadas com o objetivo de oferecerem elevada performance para aplicações *web* em escala. Fornecem estruturas de dados livres e de fácil replicação, e em alguns casos persistentes. O termo NoSQL é traduzido em várias comunidades como "não só sql", que significa que estas bases de dados não relacionais não substituem as clássicas bases de dados relacionais, existindo aplicabilidade para ambas.

Existem vários tipos de bases de dados NoSQL: *key–value stores*, *tuple store*, *object database*, *document store* e *wide columnar store*. Todas elas oferecem características diferentes de gerir dados, sendo as mais utilizadas as *key-value stores* e *document store*, como os seus nomes indicam uma é orientada a chave-valor e a outra ao documento.

As bases de dados *key-value,* são simples de usar e oferecem elevada escalabilidade, performance e flexibilidade. Isso deve-se à sua estrutura de dados simples, ao facto de algumas destas bases de dados carregarem os dados em memória física e os métodos que permitem questionar são simples, do tipo GET key.

Nos NoSQL do tipo *document oriented,* cada base de dados é uma coleção de documentos independentes, ao contrário das bases de dados relacionais que armazenam os dados em tabelas relacionadas entre si. Cada documento possui uma estrutura própria não relacional em objetos tipo xml, json, yaml entre outros tipos.

A principal vantagem destes NoSQL em comparação às clássicas bases de dados relacionais é a performance em devolver dados. Enquanto numa pesquisa a um modelo relacional, é necessário questionar uma ou mais tabelas relacionadas com JOIN's, com operações por vezes complexas, os NoSQL não têm essa complexidade, uma determinada informação encontra-se registada numa determinada chave ou documento.

Existem vários NoSQL do tipo *key-value* ou *document oriented,* para este projeto de Mestrado foram analisados os serviços couchbase, mongoDB e redis. Embora sejam diferentes tipos de servidores ambos oferecem as condições de armazenamento que se pretendia. O couchbase e o mongoBD permitem o armazenamento de documentos, oferecem elevados níveis de desempenho e escalabilidade. O redis, uma NoSQL do tipo *key-value* disponibiliza um conjunto de estruturas interessantes mantendo os elevados níveis de performance para que foi concebido, dados do tipo *hashes, lists, sets* e *sorted sets* fazem deste NoSQL um solução interessante para aplicações que se pretendem com elevado desempenho.

No anexo 1 encontra-se uma tabela detalha que compara estes três NoSQL [IT, solid, 2013] .

No contexto deste projeto de Mestrado, o autor entendeu que a utlização de um NoSQL aumentaria os níveis de performance da aplicação diminuindo os acessos à base de dados relacional. Como se prentede uma escalabilidade horizontal da aplicação, os NoSQL permitem tornar esta abordagem simples pelo facto de estes escalarem facilmente, e a comunicação entre as réplicas ser simples. A escolha para este componente tecnológico foi o redis, por:

- Oferecer maiores níveis de performance
- Estruturas de dados do tipo *Hashes e Lists*
- Permitir a pesquisa de chaves através de *wildcards*
- Disponibilizar *transactions*
- Persistência de dados assíncrona e parametrizável
- Dados permanecem em memória.

3.2.6 WebSocket's

O autor entendeu que a tecnologia mais adequada para a comunicação em tempo real na plataforma web são os WebSocket's. Embora nem todos os browsers suportem esta tecnologia, o facto de ser uma especificação integrada no HTML5, o autor acredita que a sua integração será uma realidade num curto espaço de tempo. Optou por não adotar o socket.io pela integração necessária que este exigia por parte do *web server*, e pelo facto do Tornado suportar nativamente WebSocket's.

3.2.7 Apache cordova

No contexto atual é possivel criar web sites com formatos de apresentação que podem variar entre dispositivos. Os browsers evoluíram e os das plataformas móveis não são exceção, sendo capazes de interpretar contéudos web em formato HTML5, CSS3 e com motores de renderização de Javascript muito avançados, tipo webkit.

Os conteúdos que este projeto pretendem produzir seriam perfeitamente adaptáveis a um web site otimizado para plataformas móveis. No entanto encontrar-se-iam limitadas as funcionalidades de cada browser, correndo o risco de incompatibilidades e mesmo de

funcionalidades desativadas que são necessárias, como por exemplo Javascript. O autor entendeu que o desenvolvimento de uma aplicação nativa acrescentaria maior valor à solução: maior controlo no contexto em que vai ser executada; uma redução de tráfego de dados por ser apenas necessárias chamadas à api, sem conteúdos de apresentação como imagens, html, css entre outros conteúdos; controlo de algumas funcionalidades nativas como orientação do ecrã, geolocalização, câmeras etc.

A tecnologia adotada para a criação da aplicação móvel foi o Apache cordova. Como se constatou no capitulo 2.5.1, esta tecnologia é a que suporta o maior número de plataformas móveis, é a que apresenta maior integração com as funcionalidade nativas de cada uma das plataformas, como se pode constar na tabela 5.

	android	**blackberry (6)**	**blackberry (10)**	**ios**	**WP7**	**WP8**	**firefoxos**
Accelerometer							
Camera							
Capture							
Compass							
Connection							
Contacts							
Device							
Events							
File							
Geolocation							
Globalization							
InAppBrowser							
Media							
Notification							
Splashscreen							
Storage							

Tabela 5 - Apache cordova, plataform APIs [CORDOVA, Apache]

O facto do Apache Cordova usar o HTML5 como linguagem de programação permite um rápido desenvolvimento e a integração com WebSocket's.

3.3 Infraestrutura do sistema

Um dos factores que condicionam a escalabilidade de um sistema é a sua infraestrutura física. É importante um planeamento de forma que no futuro existam condições para uma evolução robusta e tranquila.

No actual contexto, quando se fala em escalabilidade ou *cloud computing* surgem serviços como Amazon Web Services (AWS) ou Windows Azure entre outros, que disponibilizam soluções "elásticas" para escalar *hardware* e software. No caso da AWS, é oferecido um conjunto completo de serviços de computação em nuvem para permitir a criação de aplicativos sofisticáveis e escaláveis. Os serviços disponibilizados permitem um conjunto de integrações com o software desenvolvido de forma a permitir tirar o máximo partido da arquitetura.

Um critério importante na escolha de uma infraestrutura é o custo. Após uma análise a vários serviços de *cloud computing* como o já referido, a serviços de virtualização, servidores virtuais e físicos, o autor conclui que soluções de *cloud computing* como a AWS em comparação com serviços de virtualização padrão tem custos mais elevados, em alguns casos quase o dobro. É óbvio que as funcionalidades disponibilizadas também são uma váriavel que deve constar na equação, no entanto para este projeto em particular, o autor entendeu adotar uma infra estrutura de máquinas virtualizadas.

A solução adotada passa por dois servidores virtualizados, ambos podem evoluir as suas características de hardware, tendo apenas um *downtime* de um *reboot*, tempo médio de 10seg. Como se pretendeu implementar uma solução horizontalmente escalável, esta abordagem é razoável, podem ser adicionadas ou removidas máquinas mediante as necessidades, sendo estas espectáveis. O custo financeiro desta solução é inferior e as características de cada uma das máquinas é superior a uma solução equivalente em preço num serviço de *cloud computing* tipo a AWS.

Propõe-se a implementação de um servidor de Firewall e CDN externo, que permitesse filtrar todo o trafégo antes do mesmo chegar ao serviço e fazer *cache* do contéudo estático *web*.

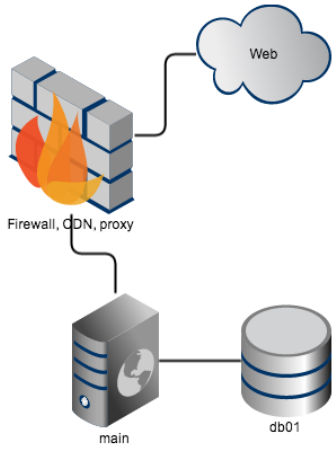


Figura 17 Nova Infraestrutura de sistema

A máquina *main* possui o servidor *web nginx* e as várias instâncias da aplicação. O servidor db01 é apenas uma máquina de *storage* com um servidor de mysql. A separação destes dois serviços permite que outras máquinas com servidores *web* surjam à medida das necessidades, mantendo o mesmo servidor de *storage.* Mesmo que seja necessário aumentar o processamento da máquina de storage o mesmo é possível sem afetar as restantes máquinas.

4 Solução desenvolvida

O projeto de Mestrado aqui apresentado é composto por vários componentes como foi descrito no capítulo 3. O desenvolvimento destes componentes foi realizado faseadamente à medida que o desenho da solução se tornava claro e objetivo. Como os componentes Collector e Publisher seriam implementados sobre a *framework* Tornado em Python, optou-se por se criar uma estrutura comum a ambos os componentes, embora cada um tivesse a sua própria camada de dados, lógica e apresentação, o objetivo é poderem partilhar algum código comum, *helper's*, *parser's* e utilitários.

4.1 Collector

Este componente é responsável por oferecer uma interface aos dispositivos que recolhem dados. A solução desenvolvida foi a criação de um *webservice* REST que reconhece apenas pedidos GET e POST e estruturas de dados em JSON ou XML. A cada pedido efetuado a esta API é possível indicar no cabeçalho qual o tipo de resposta, JSON ou XML.

Por omissão o pedido devolvido é sempre JSON.

```
$ curl -v --header 'response-type: json' http://api.stm.pt
* About to connect() to api.stm.pt port 80 (#0)
> GET / HTTP/1.1
> Host: api.stm.pt
> Accept: */*
> response-type: json
>
< HTTP/1.1 200 OK
< Content-Type: application/json
{"status": 500, "result": "nothing to do!"}
```

No caso dos dispositivos que apenas suportam XML.

```
$ curl -v --header 'response-type: xml' http://api.stm.pt
* About to connect() to api.stm.pt port 80 (#0)
> GET / HTTP/1.1
> Host: api.stm.pt
> Accept: */*
```

```
> response-type: xml
>
< HTTP/1.1 200 OK
< Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <status>500</status>
  <result>nothing to do!</result>
</root>
```

Todas as respostas são compostas por dois elementos:

- **Result**, o resultado do pedido efetuado.
- **Status**, que é um código baseado nos códigos HTTP[2] onde neste contexto indica:
  - 200 – OK
  - 400 – Bad Request
  - 401 – Unauthorized
  - 403 – Forbidden
  - 500 - Internal Server Error

Para responder às necessidades que os dispositivos apresentavam implementou-se os seguintes métodos:

- */auth/(ID_TERMINAL) [POST]* – Este método não existia na anterior versão. Foi implementado com o objetivo de adicionar um nível de segurança a toda a API.
  Este método foi implementado baseado numa chave de autenticação, que é enviada através do cabeçalho do pedido com o nome de auth-access. O valor desta chave é "mascarado" de forma a evitar acessos diretos ao servidor, o valor é um sha1 de um md5 do valor real.
  Quando a autenticação é bem-sucedida o *result* devolvido é composto por um objeto com duas chaves, *auth1* e *auth2*, chaves únicas que garantem uma sessão por um período de tempo especificado na configuração da aplicação. Estas sessões são guardadas em base de dados e o seu período de tempo é renovado a cada interação com a aplicação.
  Todos os métodos que se seguem necessitam de autenticação, e para não receberem um status de 401, o seu pedido deve conter no cabeçalho os valores de auth1 e auth2.
- */logout [POST]* – Permite terminar a sessão enviada no cabeçalho do pedido. Internamente é atualizada a sessão na base de dados para um estado que não possa voltar a ser usada.
- */events/(ID_EVENT) [GET|POST]* – O método GET retorna a informação de todos os eventos e quando é enviado o ID_EVENT apenas é enviado o evento solicitado.
  O método POST, permite adicionar e atualizar eventos.

[2] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

- */competitors/(ID_EVENT) [GET|POST]* – O método GET permite obter todos os concorrentes para um determinado evento (ID_EVENT).
  O método POST, permite adicionar e atualizar concorrentes de um determinado evento.
- */record/terminal/(ID_TERMINAL) [POST]* – Adiciona novos registos RAW ao repositório.
- */record/process [POST]* – Permite registar quais os registos que já se encontram processados.
- */record/terminal/sync [POST]* – Todos os registos que ainda não foram processados, no momento em que é executado este método, são novamente enviados.

Não se pretende com este documento uma descrição exaustiva de como funciona a API, no anexo 2 encontra-se a documentação completa da API bem como o modelo de dados desta aplicação no anexo 3.

Um dos objetivos colocados era garantir que quando os dados fossem rececionados seriam imediatamente distribuídos. A solução implementada para este objetivo foi, recorrendo a biblioteca ZeroMQ, distribuir a informação através do padrão *publish–subscribe*. Os clientes subscrevem o serviço, e quando o servidor Collector recebe informação, essa é imediatamente distribuída pelos seus clientes. Neste caso, o ou os clientes são Processing Server's. Com esta abordagem, não existe uma garantia que no momento em que a informação foi distribuída exista algum cliente a tratar da mesma, podem existir problemas de comunicação que invalidem esta abordagem. É por causa deste questão que foram implementados os métodos /record/process e /record/sync, sendo possível informar o servidor sobre a informação que foi processada e solicitar a que ainda não foi processada. Existiam outras abordagens que podiam ser adotadas, uma seria através o clássico padrão *request-reply*, mas como é frequente existirem quebras de comunicação entre Collector e Processing Server, a abordagem adotada foi considerada mais adequada para o autor.

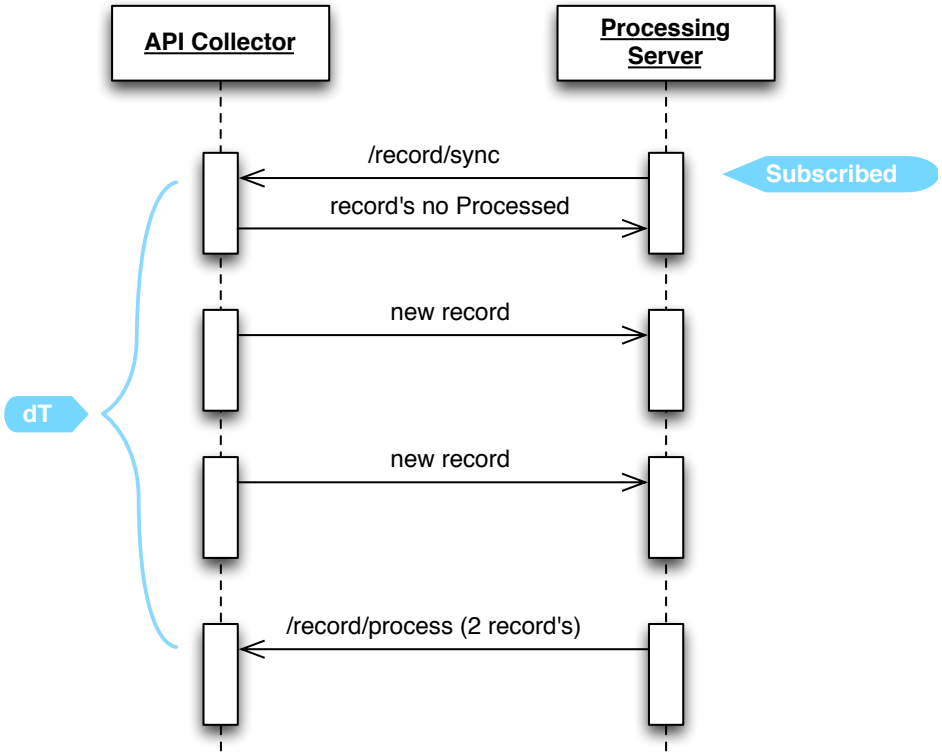


Figura 18 - Diagrama de sequência entre Collector e Processing Server

Com a implementação dos múltiplos servidores *web* Tornado e suas instâncias balanceadas pelo nginx, surgiram alguns problemas na implementação da comunicação entre Collector e Processing Server. Qual a instância que teria o serviço de Publish? Como é que as outras instâncias comunicariam com o Publish? A solução adotada foi criar um novo serviço chamado "Dispacher", este mantém o serviço de Publish num determinado porto e implementa um outro porto com o padrão *request-reply* onde todas as instâncias de servidores Tornado se ligam para entregar registos.

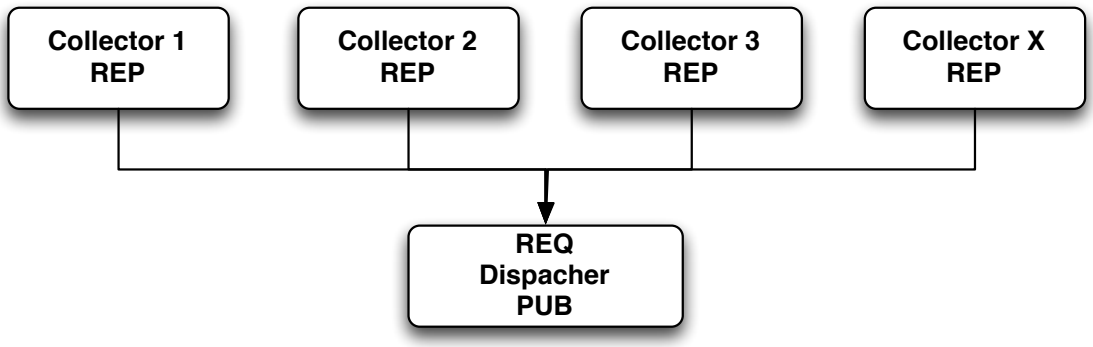


Figura 19 - Ligação do Collector ao Dispacher

Deste modo é possível criar as instâncias necessárias do Collector e um serviço de distribuição ou várias instâncias e vários pontos de distribuição distinguindo-os apenas o porto de subscrição.

```
$python3 collection/py/httpd.py 8001 127.0.0.1:10000
$python3 collection/py/httpd.py 8002 127.0.0.1:10000
$python3 collection/py/httpd.py 8003 127.0.0.1:10000
$python3 collection/py/dispacher.py 10000 10001
```

No exemplo anterior são criadas três instâncias do servidor *web*, cada um numa porta, 8001, 8002 e 8003, todos se ligam ao serviço de Dispacher que se encontra em 127.0.0.1:10000. O serviço de Dispacher é executado na mesma máquina ficando à escuta no porto 10000 e distribui aos seus *subscribe* no porto 10001. Com esta implementação é possível ter várias instâncias em várias máquinas mantendo total transparência ao componente Processing Server que apenas precisa de saber que o porto de subscrição é o 10001, é possível também adicionar novas instâncias de Collector sem que o serviço deixe de funcionar.

4.2 Módulo de ligação ao Collector

O servidor de processamento, disponibiliza uma API que permite adicionar novos módulos. Com uma interface restrita, é possível criar módulos de *input* e *output* de dados, permitindo assim estender a aplicação central a outras funcionalidades como a que pretendemos adicionar. Como já existia um módulo que recolhia os dados diretamente do repositório, implementou-se um *Class Library* (realCronoLib.dll em C#) a ser usada pelo módulo existente ou em futuras integrações com novo Collector, mantendo mesma interação com o utilizador. Deste modo alterou-se apenas a forma de comunicação do módulo existente, que comunicava através de ligação ODBC com a base de dados, e passa-se a comunicar através de um *webservice* REST.

O realCronoLib tem dois objetivos, o primeiro, comunicar com o *webservice* do Collector e enviar dados necessários a realização de um evento. O segundo, subscrever-se ao servidor de *publish* disponibilizado no Collector, receber os dados por tratar do evento e entregar a um serviço que possa processar essa informação, neste caso o Processing Server.

Para o primeiro objetivo, a biblioteca desenvolvida disponibiliza um conjunto de métodos, implementando assim *adapter* com o *webservice* Collector.

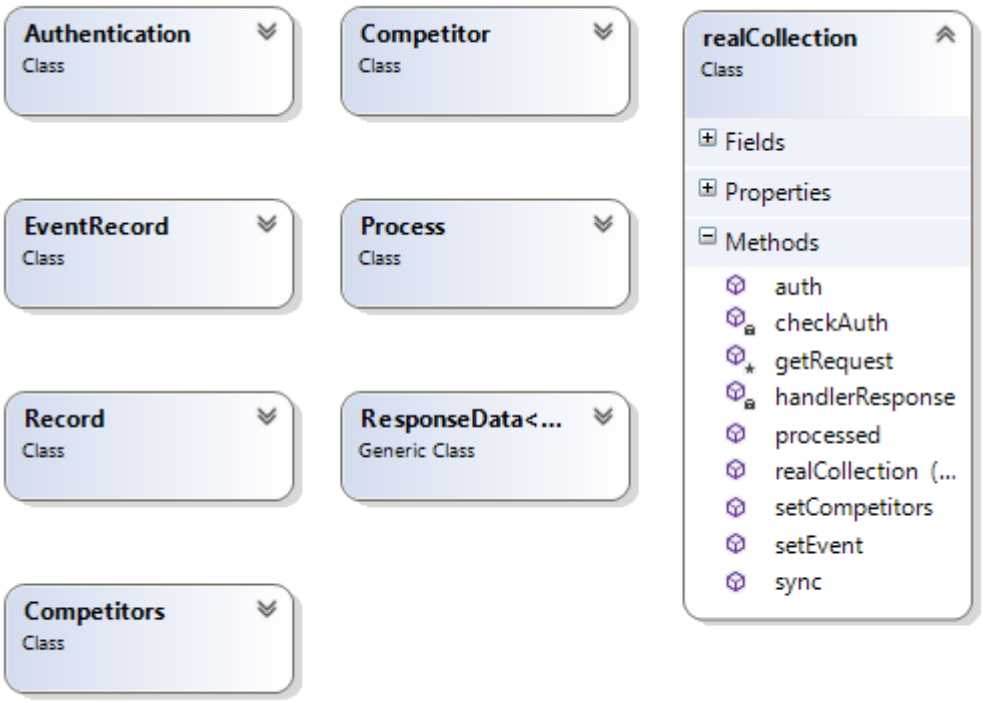


Figura 20 - Class realCronoLib

Os objetos Authentication, Competitor, EventRecord, Process, Record, ResponseData e Competitors são estruturas de dados (DataContract) que representam os objetos que a API Collector conhece. Estes classes estendem do DataContractJsonSerializer, objeto que a *framework* .net disponibiliza para serializar e deserializar estas estruturas em JSON.

A classe realCollection implementa a lógica de comunicação e autenticação com o serviço Collector. O módulo existente foi adaptado a efetuar chamadas a esta biblioteca em vez da ligação ao OBDC da base de dados remota. Esta abordagem permite abstrair o seu funcionamento do Collector, sendo apenas necessário conhecer os métodos disponíveis e os parâmetros necessários ao seu funcionamento, que por questões de integração foram mantidos de acordo com o módulo existente.

Para o segundo objetivo, foi necessário implementar um mecanismo que permitisse subscrever o serviço de *publish and subscrive* disponibilizado pelo servidor de Collector, de forma a poder receber a informação que este difunde.

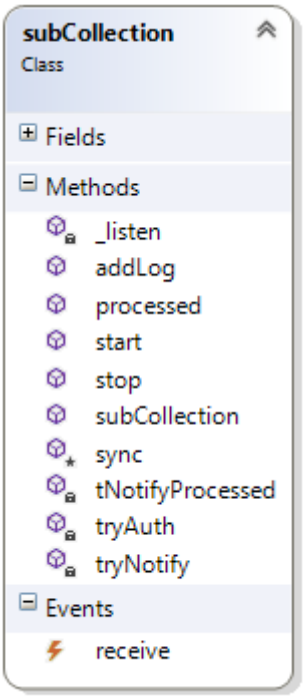


Figura 21 - Classe subCollection

Nesse sentido surge a classe subCollection, que disponibiliza um conjunto de métodos responsáveis por efetuar a subscrição e divulgação dos dados recolhidos pelo Collector. Ao executar o método *start()* o mesmo efetua uma subscrição ao serviço de *publish* e de imediato é executada uma *thread* que à medida que o servidor de Collector difunde registos, este verifica se algum objeto subscreveu o evento receive e encarrega-se de entregar estes registos ao subscritores.

Esta implementação recorreu à biblioteca clrzmq[3], uma adaptação da ZeroMQ para a *framework* .net de forma a poder subscrever o serviço disponibilizado pelo Collector que também utiliza ZeroMQ.

[3] https://github.com/zeromq/clrzmq

Para melhor compreender todo o fluxo da aplicação segue-se um diagrama de sequência.

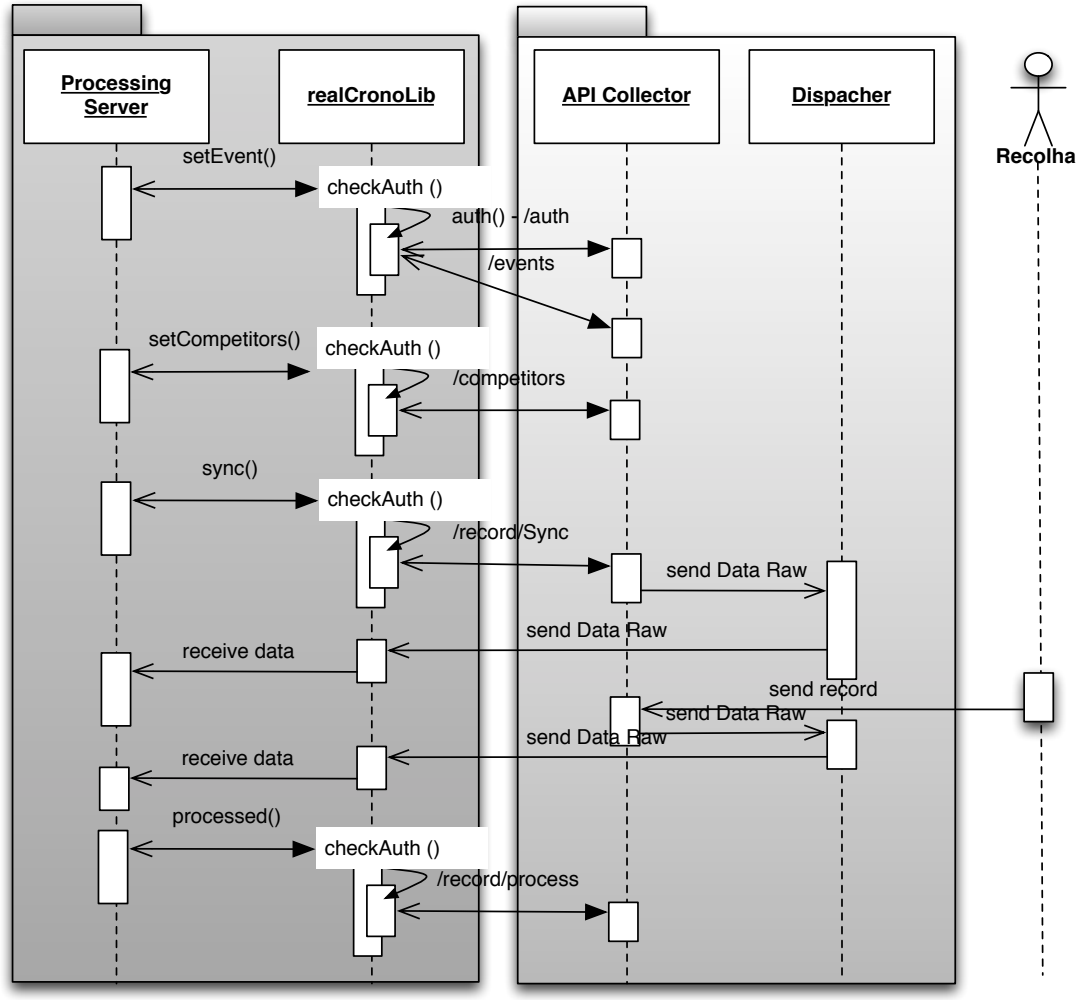


Figura 22 - Diagrama de sequência entre realCronoLib e Collector

A razão que levou o autor a adotar a implementação de uma biblioteca em vez da alteração imediata do módulo existente, foi a possibilidade da mesma poder ser integrada noutros projetos, nomeadamente software de terceiros.

4.3 Publisher

O Publisher é o componente responsável por apresentar os resultados ao público em geral. Nesta nova versão deste componente, assume um papel ainda mais relevante: passa a ser uma API que faz de interface ao Processing Server e outro serviços; e uma interface *web* com capacidade de apresentar os resultados instantaneamente sem qualquer intervenção do utilizador.

Numa primeira fase, após o levantamento de requisitos do projeto e avaliação do modelo de dados existente, verificou-se que o modelo não se encontrava totalmente normalizado e não

oferecia a estrutura necessária ao que se pretendia implementar. Após algumas tentativas de normalização dos dados existentes, optou-se por criar um novo modelo de raiz, tendo em atenção a necessidade de migrar os dados existentes. O anexo 4 representa o modelo de dados utilizado para o desenvolvimento deste componente.

De forma a tornar a aplicação mais célere e eficiente, é evitado o acesso ao disco, usando o DBMS relacional e o novo modelo apenas para a persistência de dados que são enviados pela API. Os dados apresentados aos utilizadores do serviço são todos guardados numa base de dados NoSQL (redis) do tipo *key value store*, desnormalizados e divididos em vários tipos de objetos utilizando as funcionalidades deste NoSQL.

Estes são os objetos mais relevantes guardados no redis e comuns em toda aplicação de Publisher:

Tipo	Key	Value	Variáveis
hash	sessionUser%d	Sessão de um utilizador da API	• Identificação do utilizador
string	sessionAuth%s%s	Chave de autenticação do utilizador	• Auth1 • Auth2
list	modalities	Todas os ids das modalidades suportadas	
hash	modality%d	Modalidade	• Identificação da modalidade
hash	url-%s	url's que aplicação pode resolver	• url
hash	category-%d	Categoria	• Identificação da categoria
list	categoryComp%d-%d	Concorrentes que pertencem a um categoria	• Identificação do evento • Identificação da categoria
list	competitors%d	Concorrentes pertencentes a um evento	• Identificação do evento
hash	competitor%d	Concorrente	• Identificação do concorrente
hash	event%d	Evento	• Identificação do evento
list	allevents	Lista de todos os eventos	
string	event-%s	Identificação de um evento através do seu url	• url do evento
list	eventCategories%s	Lista de categorias por evento	• Identificação do evento
hash	race%s-%d-%d-%d-%d	Resultado de uma prova de um concorrente	• Tipo de resultado • Identificação de um grupo • Identificação de uma stage

			• Categoria • Posição
string	'sircrono_%s	Identificação de uma sessão de utilizador.	• Identificação única de uma sessão.
list	stages%d	Lista de especiais de um evento	• Identificação de um evento
hash	stage%d	Dados de uma especial	• Identificaçãoo de uma stage

Tabela 6 - Chaves usadas no NoSQL

4.3.1 API Publisher

Implementou-se um *webservice* REST com o objetivo de dar resposta a dois tipos de clientes, o Processing Server e aplicações externas como por exemplo as plataformas móveis, cada um deles com nível de acesso diferentes, um permite inserir e visualizar, o outro apenas permite visualizar dados.

O protocolo estabelecido para envio e resposta de mensagens é semelhante ao da API Collector, neste caso as respostas são sempre do tipo JSON, não é possivel definir a *response-type* por não ser considerado necessário pelo autor. Apenas são permitidos métodos GET e POST. As respostas são compostas por dois elementos *result* e *status*. Os clientes que inserem dados tem que se autenticar, os restantes não necessitam dessa autenticação, podem de igual modo acederem à API, mas apenas têm acesso à visualização de dados, os mesmos que são apresentados na interface *web.*

Alguns métodos relevantes :

- */api/auth/login [POST]* – Permite efetuar autenticação na aplicação.
- */api/auth/logout/ [POST]* – Permite terminar a sessão na aplicação.
- */api/events [POST]* – Permite inserir e alterar eventos.
- */api/events/all [GET]* – Retorna todos os eventos registados.
- */api/events/(ID_EVENT) [GET]* – Retorna um evento mediante a sua identificação.
- */api/groupings [POST]* – Permite adicionar e editar grupos de categorias.
- */api/groupings/(ID_EVENT) [GET]* – Retorna os grupos de um evento.
- */api/categories [POST]* – Permite adicionar e editar categorias.
- */api/categories/(ID_CATEGORY) [GET]* – Retorna uma categoria mediante a sua identificação.
- */api/stages [POST]* – Permite inserir e alterar especiais.
- */api/stages/(ID_EVENT) [GET]* – Retorna as especiais de um evento.
- */api/splits/(ID_STAGE) [GET]* – Retorna a informação de um intermédio.
- */api/competitors [POST]* – Permite inserir e editar concorrentes.
- */api/competitors/(ID_EVENTO) [GET]* – Retorna os concorrentes de um evento.

- */api/results [POST]* – Permite inserir e editar resultados de um evento.
- */api/results/(ID_EVENTO)-(ID_GROUPING)-(ID_CATEGORY)-(ID_STAGE)[GET]*– Retorna os resultados mediante os critérios: evento, grupo, categoria e especial.

No anexo 5 é disponibilizado a documentação da API Publisher, os métodos, formato dos objetos e as respostas obtidas.

4.3.2 Aplicação Web

Foram vários os objetivos que motivaram o desenvolvimento da nova aplicação *web*, um deles foi tirar partido de uma *framework web,* neste caso a *Tornado,* de forma a tornar os desenvolvimentos futuros mais simples e estruturados.

Recorrendo aos recursos da *framework* e a forma como a mesma trata os caminhos para aplicação, começou-se por implementar um mecanismo de gestão de url, de forma a ser possível ter URL's amigavéis, melhorar o acesso aos serviços e melhorar SEO da aplicação. A framework, na fase de inicialização do serviço permite definir rotas para a aplicação e qual o objeto que vai dar resposta a um determinado pedido.

```
app = Application([
            (r"^/(?!api|assync)(.*)$", frontend.urlManager),
            (r"^/assync/(.*)$", assync.assync),
            (r"^/api/([a-zA-Z0-9\-\/]+)$", api.api)],
            **settings)
```

Através de expressões regulares ficou definido que existiam três rotas possíveis, uma para api, outra para os pedidos AJAX (assync) e o terceiro que é responsável pelos restantes caminhos.

A classe *urlManager* que vai tratar dos pedidos genéricos implementa um mecanismo simples para tratar os url's. Todos os pedidos devem ser compostos por um modelo e opcionalmente uma ação, através desse modelo o urlManager vai procurar na base de dados redis se existe alguma entrada com esse valor, caso exista trata de redirecionar para o serviço respetivo, caso contrário é devolvido ao browser um 404 – *Not found*. O serviço recebe a ação do *urlManager*, trata esse pedido, efetua o render do pedido caso necessário e devolve um status o *urlManager* igual ao usado no protocolo HTTP: 200, 404, 500 etc.

Um outro recurso da *framework* ao qual se procurou tirar proveito foi o mecanismo de *templating* que usa. Foi possível assim separar a camada de apresentação da lógica. Procurou-se dividir em algumas áreas comuns o site, por exemplo *headers, menus, footers, etc,* mantendo o mesmo código ao longo da navegação na aplicação web. Utilizou-se a mesma estrutura de dados em variadas vistas da aplicação e como a camada lógica se encontra isolada é possível usar a mesma lógica em pedidos AJAX onde não necessita de camada de apresentação.

```
<div id="stageNav">
    <ul class="nav">
```

```
        {% for stage in stages %}
        <li><a id="stage-{{stage['id_stage']}}" class="{{stage['status']}}"
href="#">
            <div    id="stage-min-{{stage['id_stage']}}"    class="op-stage-
min">{{stage['short_name']}}</div>
            <div  id="stage-max-{{stage['id_stage']}}"  class="op-stage-max"
style="display: None">{{stage['name']}}</div>
            </a>
        </li>
        {% end %}
    </ul>
</div>
<script>
    main.bindMenuStages({{len(stages)}});
</script>
```

Como se prentendia acrescentar mais algum dinamismo ao já existente, foi necessário implementar um objecto que tratasse apenas os pedidos AJAX. A classe *assync* usa a camada de lógica existente para obter os dados, tratá-los de forma a serem enviados em formato de objeto JSON, de forma a minimizar o tráfego. Foi necessário implementar do lado do cliente algo que fosse capaz de transformar estes objetos em HTML. Através das bibliotecas jQuery e underscore.js, implementou-se à semelhança do que foi feito com a *framework* Tornado um mecanismo de *templating* que substitui os elementos na DOM que devem ser atualizados.

```
<script type="text/template" id="tmplEntries">
    <% _.each(dataEntrie, function(entrie)
    { %>
    <div id="pos<%=entrie.pos%>" class="lineResult <%=entrie.modality%>">
        <div id="pos<%=entrie.pos%>-num" class="number">
              <%=entrie.number%>
        </div>
        ...
    </div>
    <% }); %>
</script>
```

A aplicação web foi desenvolvida de forma a poder ter várias instâncias a correr, à semelhança do que foi implementado no Collector, podem ser inicializados vários servidores com esta aplicação em portas diferentes e o nginx trata de balancear a carga.

```
$python3 publish/py/realHttpd.py run 9001
$python3 publish/py/realHttpd.py run 9002
...
$python3 publish/py/realHttpd.py run 9999
```

Como as sessões e os dados de *frontend* são armazenados numa base de dados redis, podendo este ser escalada através de replicação, esta implementação permite que sejam adicionados servidores ou retirados à medida das necessidades.

4.3.3 WebSocket

A implementação de websocket levantou de imediato alguns problemas, o primeiro estava relacionado com a configuração do nginx para suportar este tipo de pedidos. Embora os estudos efetuados indicassem que o mesmo suportava, pouca foi a documentação encontrada para tornar isto possível. No anexo 6 encontra a configuração implementada no servidor de produção que permite o uso de websocket's.

Um outro problema que surgiu, está relacionado com a implementação adotada para permitir escabilidade de servidores. Os websockets permitem abrir uma ligação TCP entre o cliente e o servidor. Como podemos ter vários servidores, embora com *load balance* a sua frente, é difícel saber a qual dos servidores o cliente se liga. O mesmo acontece com a API, os pedidos são balanceados entre as várias instâncias de servidores, essa informação é enviada para o serviço, a mesma deve ser enviada de imediato aos sockets abertos naquele instante. Mas quais e em que servidores estão esses sockets? A solução encontrada para ultrapassar estes problemas passou por implementar um mecanismo de *queuing*.

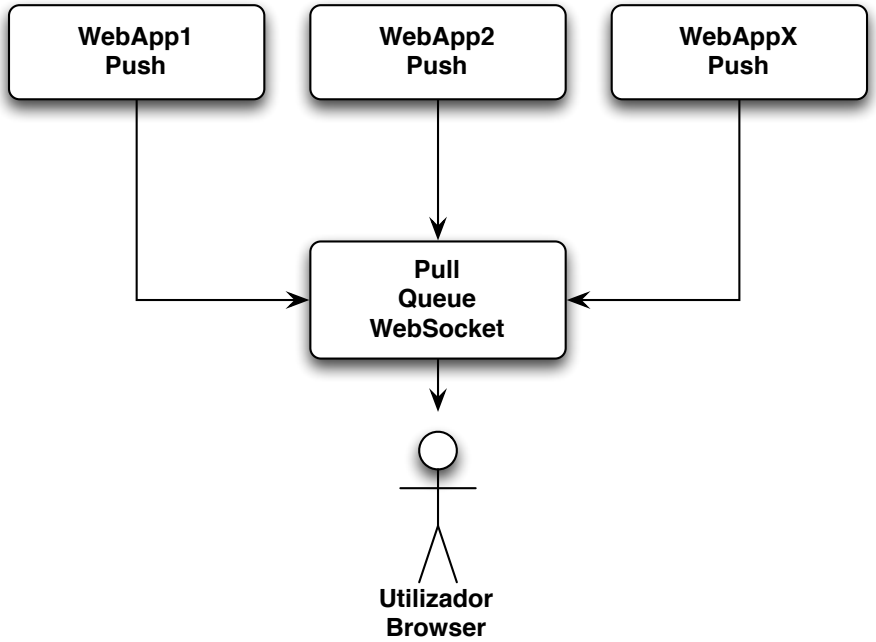


Figura 23 Fila de comunição com o servidor de websocket's

Passa a existir um serviço central de *pull* que delega aos servidores de websockets a informação a enviar.

Para implementar esta abordagem recorreu-se novamente à biblioteca ZeroMQ, que através do padrão *PUSH and PULL* é possível implementar esta solução rapidamente. Deste modo a informação chega à API seja qual for o servidor, após os dados serem guardados no redis é chamado o serviço de *push* que envia um objeto com o tipo de informação e os dados necessários que identificam a informação a enviar aos clientes.

```
{
'type': 'results',
'id_stage': 20,
'id_category': 399,
'id_grouping': 118
}
```

No lado do serviço de *PULL*, a informação recebida é imediatamente validada e verifica-se se existe algum cliente que necessita desta informação.

```
def send_data(data):
    if 'type' in data:
        if data['type'] == 'results':
            clients_visibility = [client for client in
               WebSocketService.users
                                  if {'id_stage', 'id_grouping',
                                   'id_category'} <= set(client.session)]
            if(len(clients_visibility)) > 0:
                clients = [client for client in clients_visibility
                           if int(client.session['id_category']) ==
                                    data['id_category']
                              and int(client.session['id_grouping']) ==
                                    data['id_grouping']
                              and int(client.session['id_stage']) ==
                                    data['id_stage']]
                if len(clients) > 0:
                    evt = EventLogic(data)
                    datasend = evt.get_results()
                    for client in clients:
                        client.write_message({'type': data['type'], 'data':
                                              datasend})
    else:
        print('unknown information:')
```

De forma a tornar o serviço mais eficiente o serviço de websocket apenas é disponibilizado em resultados e quando a prova se encontra num estado de *ONGOING,* isto é quando a prova se encontra a decorrer, sendo que após a conclusão os dados passam a ficar estáticos e inalteráveis.

4.4 Módulo de ligação ao Publisher

À semelhança do módulo de ligação ao Collector, foi adaptado o módulo existente de *output* para a internet, a nova API Publisher. O módulo existente enviava dados diretamente através de ODBC ao antigo servidor web, obrigando a que o servidor de processamento conhecesse detalhadamente a estrutura da base de dados existente. Para comunicar com a nova API houve necessidade de adaptar toda estrutura de dados existente a objetos JSON que são aceites na nova API. Foram adaptados os métodos que efetuavam ligações ODBC e os DataSet foram substituídos pelos novos objetos JSON:

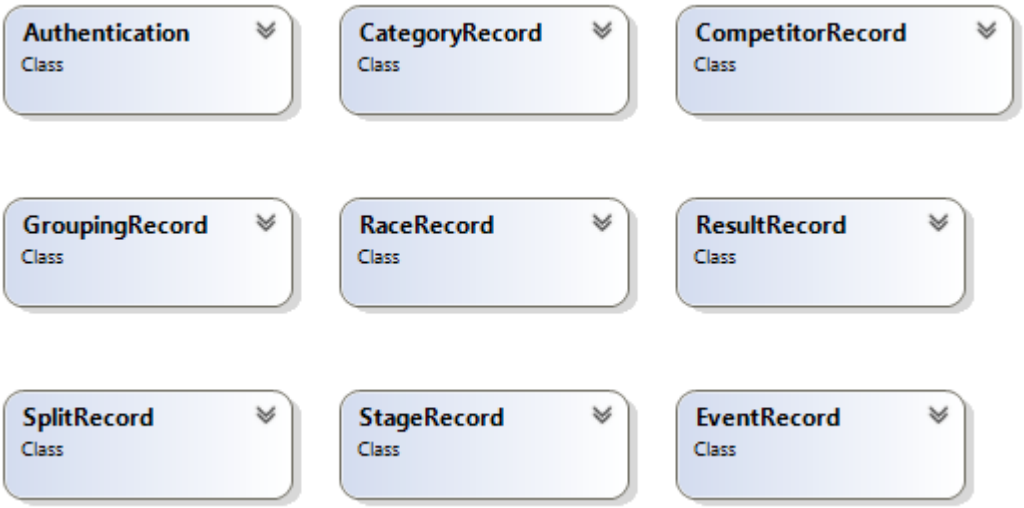


Figura 24 - Objetos usados para comunicar com o Publisher

Implementou-se um *adapter* com a API Publisher, permitindo assim substituir as antigas chamadas ao OdbcConnection a nova calsse RealPublish:

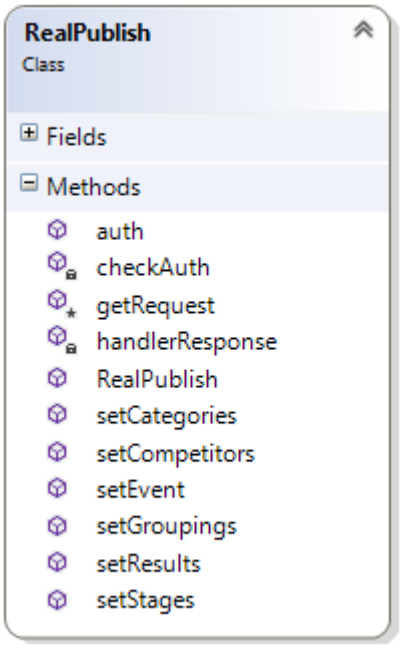


Figura 25 - Classe RealPublish

4.5 Aplicação móvel

A implementação da aplicação móvel seguiu a estrutura definida pela *framework* Apache Cordova, que cria uma estrutura de ficheiros e pastas que permite separar o código fonte, os plugins, merges e as múltiplas plataformas. No âmbito deste projeto apenas foram consideradas duas plataformas móveis android e ios.

O desenvolvimento da solução foi simples, como esperado, a framework disponibiliza um ficheiro de configuração (config.xml), onde podem ser definidas algumas diretivas para cada uma das plataformas, como icon da app, nome, descrição etc. Foram criados quatro documentos em HTML5 que funcionam como esqueleto dos ecrãs:

- events – onde é disponibilizada a informação de eventos.
- menu – de opções disponíveis para o um evento.
- competitors – detalha a informação dos concorrentes de cada evento.
- results – apresenta os resultados de uma prova.

Estes documentos são estruturas vazias de que são preenchidas por Javascript, à medida que o utilizador interage com o dispositivo, a framework lança eventos, esses eventos são tratados por Javascript mediante o gesto e o objeto que sofreu essa interação. Quando um utilizador carrega num de um evento, através de Javascript é efetuado um pedido ajax à API do Publisher, mediante a sua resposta é apresentado o menu ou uma mensagem de erro.

O Apache Cordova permite adicionar diversos plugins Javascript de forma a aumentar as suas funcionalidades. Nesse sentido foram adicionados os plugins :

- jQuery – biblioteca que permite simplificar a criação de scripts no lado do cliente.
- jQuery mobile – um plugin para jQuery que adiciona funcionalidades Touch.
- Underscore.js – uma biblioteca que disponibiliza um motor de *template* em javascript de forma a separar o código de lógica com o de visualização.

A implementação de WebSocket seguiu a mesma abordagem da versão Publisher, no entanto esta funcionalidade ainda não se encontra disponível na versão 4.2 do android, foi necessário adotar uma funcionalidade externa que permitisse acrescentar essa funcionalidade a esta plataforma.

Foi adotado o projeto websocket-android-phonegap, um projeto open-source disponível no github compatível com o Apache Cordova, permitindo acrescentar essa funcionalidade a esta plataforma. A sua integração foi simples, bastando adicionar o código android na pasta da plataforma android, e o plugin Websocket.js foi adicionado apenas a estas plataformas.

5 Validação Experimental

A solução proposta foi implementada na sua totalidade e já se encontra em produção. O primeiro evento que esta solução suportou, foi uma prova do Campeonato Nacional de Open de Ralis 2013, onde participaram 34 concorrentes, num itinerário de 143,76km sendo 63.28km em competição divididos em 6 classificativas. Este evento foi selecionado para o lançamento da solução, pelo facto de conter um número reduzido de concorrentes, minimizando assim possíveis problemas caso surgissem erros relacionados com esta nova abordagem. Era espetável um número de acessos no dia do evento aproximado de um milhar, sendo nestes casos, acessos em simultâneo devido à expetativa criada de saber quem são os vencedores de cada uma das classificativas, categorias ou campeonatos.

Utilizou-se a ferramenta GoAccess[4] para analisar o ficheiro de access.log do servidor de apenas esse evento, onde se obtiveram os seguintes resultados:

- Numero de visitas únicas no dia do evento: 1453
- Número de pedidos : 151929
- Estado dos pedidos :

Pedidos	Percentagem	Status
114528	75,38%	200
25966	17,09%	304
5712	3,76%	101
2225	1,46%	404
1525	1,00%	499
712	0,47%	500

Tabela 7 - Estado dos pedidos em produção

Verifica-se que 96% dos pedidos foram bem sucedidos, sendo 5712 (3,76%) *Switching Protocols* para WebSocket's mantendo a ligação estabelecida. Foi possivel verificar que

[4] http://goaccess.prosoftcorp.com/

durante o decorrer do evento existia uma média de 53 socket's permanentemente ligados à aplicação, sendo que o socket apenas se encontra disponível quando uma determinada classificativa se encontra a decorrer.

O tempo entre a recolha e a publicação de resultados não foi possível apurar de uma forma rigorosa. As condições de comunicação entre postos nem sempre foram as ideais, existindo alguma latência de comunicação, mesmo assim pode-se verificar que a média de publicação de resultados é inferior a 5seg, isto é, recolha no terreno do resultado, envio pelo Collector para o Processing Server e deste para o Publisher que difunde através de um websocket. Este valor embora não seja rigoroso é claramente diferente de 60seg da abordagem anterior.

Em relação à performance e escalabilidade que a nova aplicação apresenta, o evento realizado, devido ao baixo número de acesso não reflete as características pretendidas, deste modo recorreu-se a ferramenta *ab - Apache HTTP server benchmarking tool*[5], para efetuar uma bateria de testes relacionados com a performance e escalabilidade da aplicação.

As aplicações encontram-se na mesma infraestrutura de hardware. A antiga aplicação usa o nginx como *webserver* que redireciona os pedidos para o php5-fpm *FastCGI Process Manager,* a nova arquitetura usa o nginx como *reverse proxy* e *load balance* que encaminha os pedidos para quatro instâncias do serviço.

Foi implementado um *bash script* que se encontra no anexo 7, que permite executar um número de pedidos em simultâneo. A cada uma das aplicações, é executado um pedido semelhante em tamanho de resposta (25000 bytes), onde se podem verificar os seguintes resultados :

Pedidos/Simultâneo	versão PHP		versão python+tornado	
	(t ms)	NOT_2XX	(t ms)	NOT_2XX
5000/1	265,7	0	280,3	0
5000/2	261,4	0	304,0	0
5000/4	265,3	0	326,1	0
5000/8	270,8	0	406,1	0
5000/16	292,7	0	549,9	0
5000/32	379,6	0	1088,1	0
5000/64	638,8	0	2120,8	0
5000/128	1241,1	1402	4364,3	0
5000/256	1684,1	2039	8470,0	0
5000/512	1314,8	3701	17912,4	11

Tabela 8 - Comparativo de tempo de resposta entre as aplicações

[5] http://httpd.apache.org/docs/2.2/programs/ab.html

Foram executados 10 testes, cada um com 5000 pedidos variando apenas o número de pedidos em simultâneo (2^x).

Pode verificar-se que quando a carga de utilização é baixa a anterior solução apresenta um desempenho superior à nova solução. A explicação para isto acontecer, está relacionado com o facto da antiga versão ser em PHP scripting sem qualquer framework ou mecanismos de *templating* oferecendo uma interface básica com pouca interação.

Na nova aplicação foram implementados um conjunto de tecnologias que como é óbvio prejudicam a performance beneficiando a interação com o utilizador.

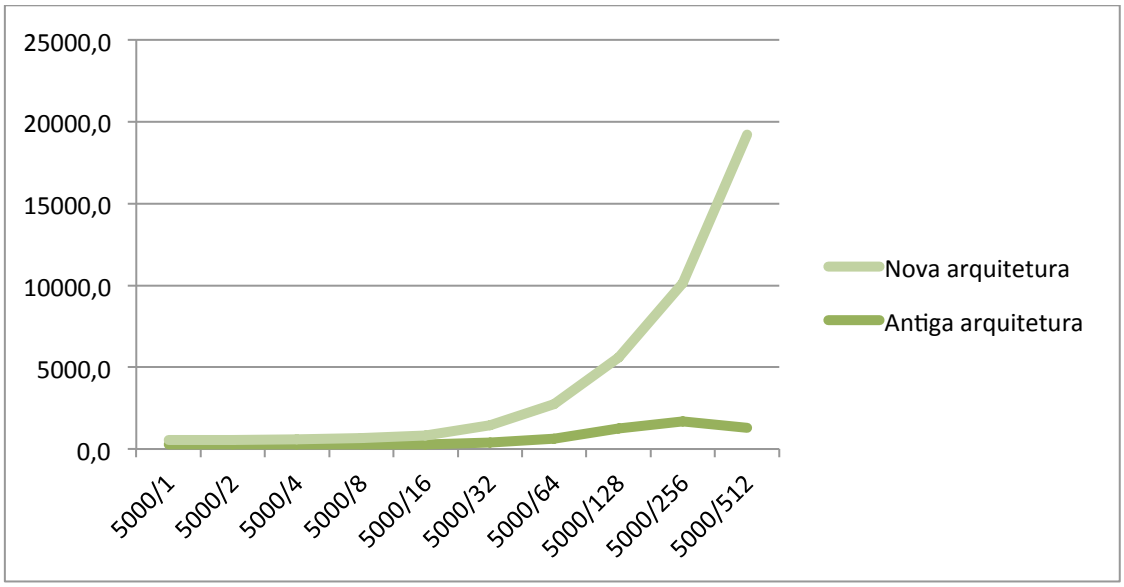


Figura 26 - Tempo de resposta entre arquiteturas

Verifica-se também que, a antiga versão a partir de um determinado número de pedidos em simultâneo começa a rejeitar pedidos, devolvendo um *status 500,* ao utlizador.

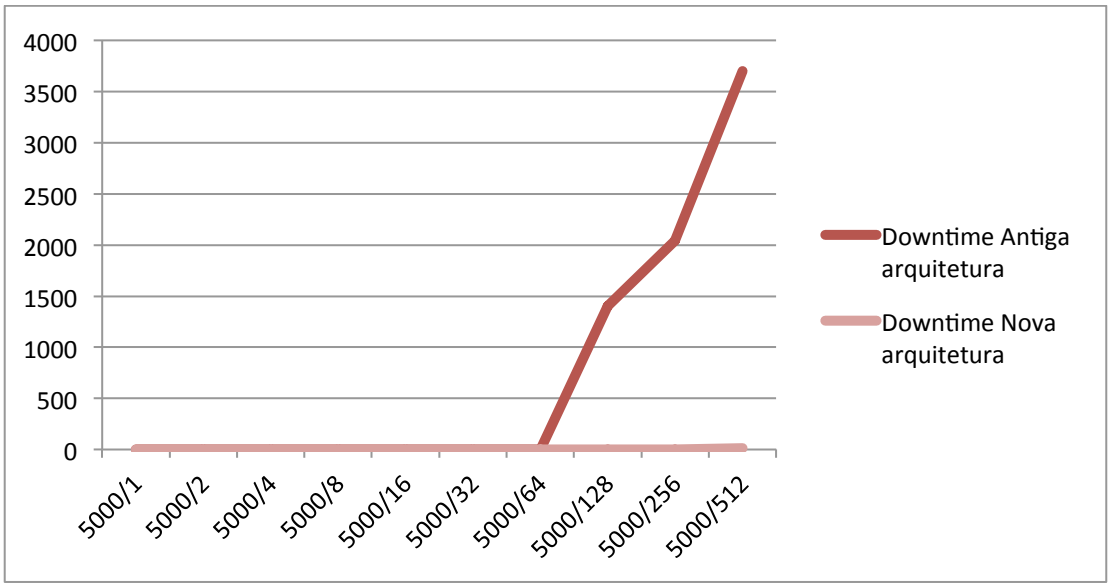


Figura 27 - Pedidos recusados (Downtime)

À medida que o número de pedidos em simultâneo chegam ao servidor, vários pedidos são rejeitados, provocando um *downtime* do serviço. Verificou-se nesse período de tempo que o servidor de mysql deixou de aceitar ligações, ficando mesmo inacessível até reiniciar o serviço.

Os resultados aqui apresentados demonstram que a atual solução é capaz de suportar mais carga que a antiga, no entanto também se verifica que existe uma perda de performace a qual não era previsível. Isso devesse a vários factores: a infraestrutura com características limitadas de hardware; alguns algoritmos de visualização que foram adicionados provocando maior processamento a cada pedido; um fraco aproveitamento da framework devido a inexperiência do autor e o aumento de serviços que agora são disponibilizados.

Esta nova abordagem apresenta maior complexidade de processamento do que a anterior, isso acontece pelas novas funcionalidades e serviços disponibilizados. A nova aquitetura prevê que a mesma seja executada em várias máquinas em simultâneo aumentando a capacidade de carga e de desempenho. Esse cenário não foi possivel testar, no entanto face aos resultados aqui apurados é possivel prever que os resultados sejam satisfatorios.

6 Conclusões

O projeto de Mestrado realizado iniciou-se com o estudo sobre a escalabilidade e performance de aplicações *web*, nomeadamente os diferentes tipos e quais os factores que influenciam a escalabilidade de uma aplicação *web*. Analisaram-se algumas técnicas como escalabilidade horizontal, *load balances* e mecanismos de *caching*. Procurou-se também analisar alguns serviços de *message queuing* para serem adotados como instrumento de comunicação entre a aplicação.

Ainda nesta fase foram estudadas as abordagens de comunicação em tempo real numa plataforma *web*, no entanto, de acordo com os objetivos deste projeto foi dada maior atenção à tecnologia WebSocket introduzida pelo HTML5. Esta análise revelou que esta seria a tecnologia ideal para a solução que se pretendia para este projecto.

Foram também analisadas *framework's* de desenvolvimento para plataformas móveis, que permitiu de entre várias abordagens escolher o Apache Cordova para o projeto realizado.

A solução proposta passou por uma restruturação de todos os serviços web existentes, adotando uma nova arquitetura baseada em SOA, permitindo assim a introdução de plataformas móveis ao sistema.

No contexto atual, a oferta tecnológica é alargada, para o mesmo problema existem várias soluções, tornando o processo de escolha em algo complexo e por vezes problemático. Este foi um dos maiores desafios deste projeto. Tratando-se de um sistema distribuido com vários componentes a sua implementação exigiu um elevado número de tecnologias que o autor desconhecia.

A adoção de Python como linguagem de programação e a *framework* Tornado acrescentou a este projeto um risco que o autor quis assumir face as expectativas que tinha em relação a estas tecnologias. Esta decisão foi baseada em artigos, *benchmark's* e *case studies*. Embora os resultados apresentados são claramente positivos, o autor esperava resultados ainda mais diferenciadores.

A escalabilidade horizontal adotada neste projeto acrescentou maior flexibilidade à solução, tornando possível a replicação do serviço em vários servidores sem que isso afetasse o seu

funcionamento. Embora não tenha sido possivel testar a solução em multíplos servidores, o autor está certo que os resultados serão positivos.

A utilização de NoSQL em vez de uma base de dados relacional permitiu aumentar a disponibilidade da aplicação, como se pode verificar nos resultados apurados. O principal *bottleneck* da antiga arquitetura era o acesso a dados, provocando *downtime* do serviço a partir de um número de acessos simultâneos.

As várias API disponibilizadas, permitiram acrescentar um nível de segurança inexistente e principalmente aumentar a interoperabilidade do sistema com serviços externos como plataformas móveis, entre outros.

Os WebSocket's acrescentaram um experiência de tempo real inexistente. Foi possível comprovar pela opinião dos utilizadores do serviço, que esta tecnologia acrescenta maior valor ao serviço, sendo possível obter os resultados aferidos antes deles serem tornados públicos por qualquer outro sistema.

A introdução da aplicação móvel, para além de reduzir o trafégo de dados com o servidor, permite disponibilizar o serviço onde ele mais é requerido que é no local da prova. Os principais utilizadores deste serviço são concorrentes e aficionados pelo desporto em causa, que normalmente se deslocam ao local do evento, sem acesso a um computador mas certamente com um dispositivo móvel.

As principais dificuldades encontradas ao longo deste projeto estiveram relacionadas com a diversidade de tecnologias adotadas. A experiência em Python foi muito positiva em termos de desenvolvimento, mas pouco satisfátoria em termos de performance. O ORM adotado tornou-se um problema ao desenvolvimento da aplicação, com a sua introdução pretendia-se que o processo de persistência de dados fosse abstraido pelo ORM permitindo apenas manipular objetos, embora tenha sido isso que aconteceu o tempo de adaptação a estes objetos foi elevado perdendo-se assim o objetivo de acelarar o tempo de desenvolvimento.

6.1 Trabalho Futuro

Os objetivos estabelecidos para este projeto foram cumpridos, no entanto existe um longo caminho a percorrer com esta nova arquitetura. Embora a aplicação já se encontre em produção existem algumas otimizações que ficaram pendentes e novas funcionalidades que não fazendo parte deste projeto faz todo o sentido existirem.

- Aumentar os contéudos em cache de forma a evitar cálculos constantes a pedidos repetidos, melhorando assim a performance da aplicação.
- Aumentar as funcionalidades disponíveis na aplicação móvel, como por exemplo geolocalização de resultados, partilha de imagens dos eventos integrado na aplicação.
- Evoluir a aplicação para dois componentes apenas, Collector e Publisher evoluindo o Processing Server para a cloud.

- Melhorar a usabilidade da aplicação web, criando um portal de resultados.
- Criar integração com as redes sociais.

7 Anexos

1. Tabela de comparação de NoSQL
2. Documentação sobre a API Collector
3. Modelo de dados Collector
4. Modelo de dados Publisher
5. Documentação sobre a API Publisher
6. Configuração nginx para WebSocket's
7. Bash Script de testes da aplicação
8. Resultados dos testes efetuados

Referências

ADLER, Brian. 2011. Building Scalable Applications In the Cloud.

BONDI, André B. 2000. Characteristics of scalability and their impact on performance.

Can I use, support tables for HTML5, CSS3. 2013. [online]. Available from World Wide Web: <http://caniuse.com/>

CARBOU, Mathieu. 2011. Reverse Ajax. *Developer Works*.

CORDOVA, Apache. *Apache Cordova API Documentation*. [online]. Available from World Wide Web: <http://cordova.apache.org>

Evolution of COMET. 2008. [online]. Available from World Wide Web: <http://vinaytech.wordpress.com/2008/11/18/evolution-of-comet/>

HINTJENS, Pieter. 2013. *Code Connected - Learning ZeroMQ*. iMatix Corporation.

IT, solid. 2013. *Knowledge Base of Relational and NoSQL Database Management Systems*. [online]. Available from World Wide Web: <http://db-engines.com/>

LIU, Henry H. 2009. *Software Performance and Scalability*. Wiley.

MATSUDAIRA, Kate. 2012. *Scalable Web Architecture and Distributed Systems*. [online]. Available from World Wide Web: <http://www.aosabook.org/en/distsys.html>

NEUMAN, B. Clifford. 1994. *Scale in Distributed Systems*.

PETER , Lubbers, Albers BRIAN, and Salim FRANK. 2010. *Pro HTML5 Programming*.

SALVAN, Muriel. 2013. *A quick message queue benchmark*. [online]. Available from World Wide Web: <http://x-aeon.com/wp/2013/04/10/a-quick-message-queue-benchmark-activemq-rabbitmq-hornetq-qpid-apollo/>

SQLALCHEMY. 2013. *SQLAlchemy - The Database Toolkit for Python*. [online]. Available from World Wide Web: <http://www.sqlalchemy.org/>

TEQLOG. *Message Oriented Middleware: QPID ActiveMQ RabbitMQ OpenAMQ ZeroMQ*. [online]. Available from World Wide Web: <http://www.teqlog.com/message-oriented-middleware.html>

VOXEO. *http://voxeo.com/*. [online]. [Accessed 2013]. Available from World Wide Web: <http://voxeo.com/glossary/what-is-webrtc/>

Web Services Architecture. 2004. [online]. Available from World Wide Web: <http://www.w3.org/TR/ws-arch>

WEBRTC. 2013. *General Overview - WebRTC*. [online]. Available from World Wide Web: <http://www.webrtc.org/reference/architecture>

Anexo 1

Name	Couchbase	MongoDB	Redis
Description	JSON-based document store derived from CouchDB with aMemcached-	One of the most popular document stores	In-memory database with configurable options performance vs. persistency
DB-Engines Ranking	Rank : 29	Rank : 6	Rank : 13
Trend Chart	Score: 7.32	Score: 149.48	Score : 36.01
Website	www.couchbase.com	www.mongodb.org	redis.io
Technical documentation	www.couchbase.com/-docs	www.mongodb.org/-display/-DOCS/-Home	redis.io/-documentation
Developer	Couchbase, Inc.	MongoDB, Inc	Salvatore Sanfilippo
Initial release	2011	2009	2009
License	Open Source	Open Source	Open Source
Implementation	C, C++ and Erlang	C++	C
Server operating	Linux	Linux	BSD
	OS X	OS X	Linux
	Windows	Solaris	OS X
		Windows	Windows
Database model	Document store	Document store	Key-value store
Data scheme	schema-free		
Typing	no	yes	no
Secondary indexes	yes	yes	no
SQL	no	no	no
APIs and other access methods	Memcached protocol	proprietary protocol using JSON	proprietary protocol
	RESTful HTTP API		

Supported programming languages	.Net C Clojure Erlang Go Java JavaScript Perl PHP Python Ruby Scala Tcl	Actionscript C C# C++ Clojure ColdFusion D Dart Delphi Erlang Go Groovy Haskell Java JavaScript Lisp Lua MatLab Perl PHP PowerShell Prolog Python R Ruby Scala Smalltalk	C C# C++ Clojure Dart Erlang Go Haskell Java JavaScript Lisp Lua Objective-C Perl PHP Python Ruby Scala Smalltalk Tcl
Server-side scripts	View functions in	JavaScript	Lua
Triggers	yes	no	no
Partitioning	Sharding	Sharding	none
Replication methods	Master-master	Master-slave	Master-slave
	Master-slave		
MapReduce	yes	yes	no
Consistency	Eventual Consistency	Eventual Consistency	
	Immediate Consistency	Immediate Consistency	
Foreign keys	no	no	no
Transaction	no	no	optimistic locking
Concurrency	yes	yes	yes
Durability	yes	yes	yes
User concepts	simple password-based access control per	Users can be defined with full access or read-	very simple password-based access control

Specific characteristics			Redis very much emphasize performance. In any design decisions performance has
Typical application scenarios			Applications that can hold all data in memory, and that have high performance

Anexo 2

API - Collection

Interface com realCron backEnd.

São aceites apenas pedidos POST/GET com os seguintes headers opcionais.

- response-type - [xml|**json**] Formato para o qual as respostas são enviadas.
- auth1, auth2 - Obrigatório quando autenticado.

Autenticação

```
http[s]://realcron/auth/(IdTerminal)header'sauth-access : TokenID - Token de acesso aplicaçã
```

Request

- **TokenID** - Chave de autenticação composta por 64 caracteres encriptados.
- **IdTerminal** - Identificação do terminal no sistema.

Response

```
{    "status": 200,    "result": {        "auth2": "zjjpun15ls8ostdtxx19ap5m36ff53xe",
```

- **Result** - contem um **result** com **auth1, auth2** necessários para pedidos a API.

Logout

Método que permite fazer logout de uma sessão.

```
auth1auth2http[s]://realcron/logout/method : POST
```

Response

```
{    "status": 200,    "result": "sucess"}
```

Eventos

Método que permite adicionar ou alterar um **evento**.

```
auth1auth2http[s]://realcron/events/method : POST
```

Request

Para tal deve ser enviado um objeto do tipo json/xml com os seguintes campos:

- **id_event** - (Obrigatório, Inteiro maior que 0) Identifica o evento de uma forma única.
- **name** - (Obrigatório, Texto 100 caracteres) Nome ou descrição do evento.

Response

```
{    "status": 200,    "result": "sucess"}
```

Método que retorna todos os eventos disponíveis para a conta que está autenticada.

```
auth1auth2http[s]://realcron/events/method : GET
```

Request

Response

```
{    "status": 200,    "result": "sucess"}
```

Método que retorna um evento específico

```
auth1auth2http[s]://realcron/events/(ID_EVENT)method : GET
```

Request

- **ID_EVENT** - Identificação do evento.

Response

```
{    "status": 200,    "result": "sucess"}
```

Concorrentes

Método que permite adicionar ou alterar os concorrentes de um determinado evento. Este método substitui todos os dados referentes ao concorrente naquele evento.

```
auth1auth2http[s]://realcron/competitors/method : POST
```

Request

Para tal deve ser enviado um objeto do tipo json/xml com os seguintes campos:

- id_event (Obrigatório, Inteiro maior que 0) identifica o evento.
- competitors Lista de concorrentes
    - id_competitor (Obrigatório, Inteiro maior que 0) identifica o concorrente de uma forma única.
    - id_extra1 (Texto maximo de 100 caracteres) Identificação opcional
    - number (Obrigatório, Número) identificação da competição
    - short_name (Texto maximo de 50 caracteres) Abreviatura para o concorrente
    - name1 (Texto maximo de 255 caracteres) 1º Nome quando aplicável.
    - name2 (Texto maximo de 255 caracteres) 2º Nome quando aplicável.
    - name3 (Texto maximo de 255 caracteres) 3º Nome quando aplicável.
    - name4 (Texto maximo de 255 caracteres) 4º Nome quando aplicável.

- name5 (Texto maximo de 255 caracteres) 5º Nome quando aplicável.

Response

```
{    "status": 200,    "result": "sucess"}
```

Método que retorna todos os concorrentes de um determinado evento.

```
auth1auth2http[s]://realcron/competitors/(ID_EVENT)method : GET
```

Request

- **ID_EVENT** - Identificação do evento.

Response

```
{    "status": 200,    "result": "sucess"}
```

Registos

Permite registar dados.

```
auth1auth2http[s]://realcron/record/[terminal/(IdTerminal)]method : **POST**
```

Request

- **IdTerminal** - Parâmetro opcional, quando não enviado é assumido o valor de sessão.

No corpo do pedido deve ser enviado um objeto do tipo json/xml com os seguintes campos:

- type (Obrigatório) Tipo de recolha, (G1, CP505, CP520, CP540)
- id_event Identificação do evento a que corresponde este evento.
- value Valor recolhido.

Response

```
{    "status": 200,    "result": "sucess"}
```

Process - Permite notificar a API que um ou vários *records* já se encontram processados

```
auth1auth2http[s]://realcron/record/processmethod : **POST**
```

Request

No corpo do pedido deve ser enviado um objeto do tipo json/xml com os seguintes campos:

- processed O seu valor deve ser um array com todos os records já processados.

Response

```
{    "status": 200,    "result": "sucess"}
```

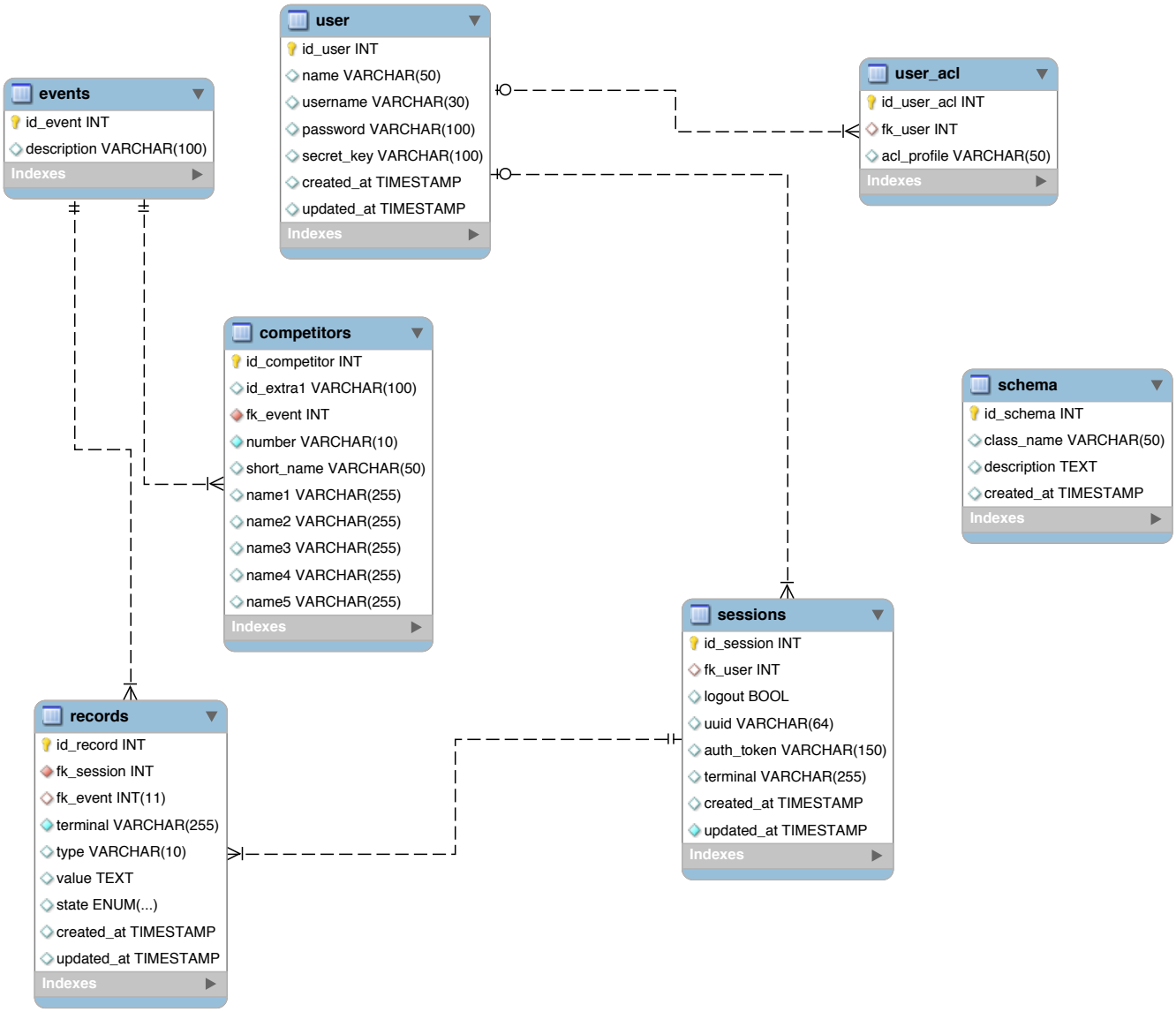
Sincronização - Envia todos os records que ainda se encontram no estado INPROGRESS.

```
auth1auth2http[s]://realcron/record/syncmethod : **POST**
```

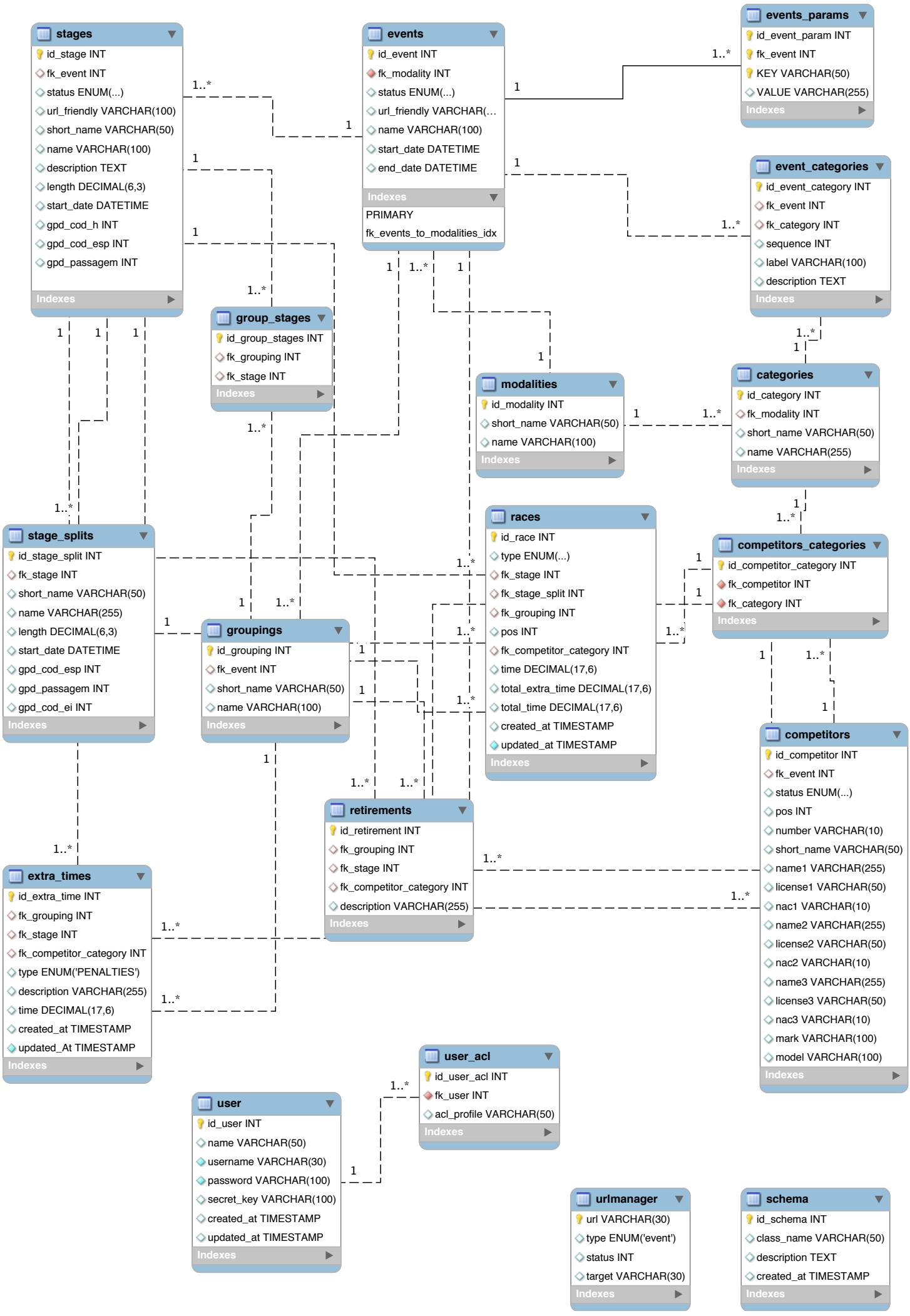
Response

```
{    "status": 200,    "result": "sucess"}
```

Anexo 3


user
id_user INT
name VARCHAR(50)
username VARCHAR(30)
password VARCHAR(100)
secret_key VARCHAR(100)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
user_acl
id_user_acl INT
fk_user INT
acl_profile VARCHAR(50)
Indexes
events
id_event INT
description VARCHAR(100)
Indexes
competitors
id_competitor INT
id_extra1 VARCHAR(100)
fk_event INT
number VARCHAR(10)
short_name VARCHAR(50)
name1 VARCHAR(255)
name2 VARCHAR(255)
name3 VARCHAR(255)
name4 VARCHAR(255)
name5 VARCHAR(255)
Indexes
schema
id_schema INT
class_name VARCHAR(50)
description TEXT
created_at TIMESTAMP
Indexes
sessions
id_session INT
fk_user INT
logout BOOL
uuid VARCHAR(64)
auth_token VARCHAR(150)
terminal VARCHAR(255)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
records
id_record INT
fk_session INT
fk_event INT(11)
terminal VARCHAR(255)
type VARCHAR(10)
value TEXT
state ENUM(...)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes

Anexo 4


stages
id_stage INT
fk_event INT
status ENUM(...)
url_friendly VARCHAR(100)
short_name VARCHAR(50)
name VARCHAR(100)
description TEXT
length DECIMAL(6,3)
start_date DATETIME
gpd_cod_h INT
gpd_cod_esp INT
gpd_passagem INT
Indexes
events
id_event INT
fk_modality INT
status ENUM(...)
url_friendly VARCHAR(...
name VARCHAR(100)
start_date DATETIME
end_date DATETIME
Indexes
PRIMARY
fk_events_to_modalities_idx
events_params
id_event_param INT
fk_event INT
KEY VARCHAR(50)
VALUE VARCHAR(255)
Indexes
event_categories
id_event_category INT
fk_event INT
fk_category INT
sequence INT
label VARCHAR(100)
description TEXT
Indexes
group_stages
id_group_stages INT
fk_grouping INT
fk_stage INT
Indexes
modalities
id_modality INT
short_name VARCHAR(50)
name VARCHAR(100)
Indexes
categories
id_category INT
fk_modality INT
short_name VARCHAR(50)
name VARCHAR(255)
Indexes
stage_splits
id_stage_split INT
fk_stage INT
short_name VARCHAR(50)
name VARCHAR(255)
length DECIMAL(6,3)
start_date DATETIME
gpd_cod_esp INT
gpd_passagem INT
gpd_cod_ei INT
Indexes
groupings
id_grouping INT
fk_event INT
short_name VARCHAR(50)
name VARCHAR(100)
Indexes
races
id_race INT
type ENUM(...)
fk_stage INT
fk_stage_split INT
fk_grouping INT
pos INT
fk_competitor_category INT
time DECIMAL(17,6)
total_extra_time DECIMAL(17,6)
total_time DECIMAL(17,6)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
competitors_categories
id_competitor_category INT
fk_competitor INT
fk_category INT
Indexes
competitors
id_competitor INT
fk_event INT
status ENUM(...)
pos INT
number VARCHAR(10)
short_name VARCHAR(50)
name1 VARCHAR(255)
license1 VARCHAR(50)
nac1 VARCHAR(10)
name2 VARCHAR(255)
license2 VARCHAR(50)
nac2 VARCHAR(10)
name3 VARCHAR(255)
license3 VARCHAR(50)
nac3 VARCHAR(10)
mark VARCHAR(100)
model VARCHAR(100)
Indexes
retirements
id_retirement INT
fk_grouping INT
fk_stage INT
fk_competitor_category INT
description VARCHAR(255)
Indexes
extra_times
id_extra_time INT
fk_grouping INT
fk_stage INT
fk_competitor_category INT
type ENUM('PENALTIES')
description VARCHAR(255)
time DECIMAL(17,6)
created_at TIMESTAMP
updated_At TIMESTAMP
Indexes
user_acl
id_user_acl INT
fk_user INT
acl_profile VARCHAR(50)
Indexes
user
id_user INT
name VARCHAR(50)
username VARCHAR(30)
password VARCHAR(100)
secret_key VARCHAR(100)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
urlmanager
url VARCHAR(30)
type ENUM('event')
status INT
target VARCHAR(30)
Indexes
schema
id_schema INT
class_name VARCHAR(50)
description TEXT
created_at TIMESTAMP
Indexes

Anexo 5

API - Pública

Interface com realCron pública.

São aceites apenas pedidos POST/GET com os seguintes headers opcionais.

- auth1, auth2 - Obrigatório quando autenticado.

Todos os pedidos têm uma resposta em json.

Autenticação

login

```
http[s]://rpublish/api/auth/loginheader'sauth-access : TokenID - Token de acesso aplicaçãomethod : POST
```

Request

- **TokenID** - Chave de autenticação composta por 64 caracteres encriptados.

Response

```
{    "status": 200,    "result": {        "auth2": "zjjpun15ls8ostdtxx19ap5m36ff53xe",        "auth1": "3b74e9c3eb70440fa0b40972l
```

- **Result** - contem um **result** com **auth1, auth2** necessários para pedidos a API.

Logout

Método que permite fazer logout de uma sessão.

```
auth1auth2http[s]://rpublish/api/logout/method : POST
```

Response

```
{    "status": 200,    "result": "sucess"}
```

Eventos

Método que permite adicionar ou alterar um **evento**.

```
auth1auth2http[s]://rpublish/api/eventsmethod : POST
```

Request

Para tal deve ser enviado um objeto do tipo json com os seguintes campos:

- **id_event** - (Obrigatório, Inteiro maior que 0) Identifica o evento de uma forma única.
- **modality** - (Obrigatório para um novo evento, Texto 50 caracteres) Abreviatura da modalidade.
- **status** - (Obrigatório para um novo evento, [performing|ongoing|completed]) Estado do evento.
- **url_friendly** - (Obrigatório para um novo evento, Texto 100 caracteres) url de acesso rápido ao evento.
- **name** - (Obrigatório para um novo evento, Texto 100 caracteres) Nome ou descrição do evento.
- **start_date** - (Obrigatório para um novo evento, Data (YYYY-MM-DD) *ISO 8601*) Data de inicio do evento.
- **end_date** - (Obrigatório, para um novo evento, Data (YYYY-MM-DD) *ISO 8601*)) Data final do evento.

Response

```
{    "status": 200,    "result": "sucess"}
```

Modalidades suportadas e identificação para a API.

id	description
ry	Rallye
ed	Enduro
ttm	TT Motos

cc	Circuito
tta	TT Auto
mt	Montanha

Método que retorna todos os eventos disponíveis para a conta que está autenticada.

```
auth1auth2http[s]://rpublish/api/events/allmethod : GET
```

Request

Response

```
{    "status": 200,    "result": [        {            "status": "performing",            "start": "2013-09-01",            "end"
```

Método que retorna um evento específico.

```
auth1auth2http[s]://rpublish/api/events/(ID_EVENT)method : GET
```

Request

- **ID_EVENT** - Identificação do evento.

Response

```
{    "status": 200,    "result": {        "status": "performing",        "start": "2013-09-01",        "end": "2013-09-01",
```

Groupings

Método que permite adicionar ou alterar um **Groupings**.

```
auth1auth2http[s]://rpublish/api/groupingsmethod : POST
```

Request

Deve ser enviado um objeto do tipo json composto por um array de objetos com os seguintes campos:

- **id_grouping** - (Obrigatório, Inteiro maior que 0) Identifica o grupo de uma forma única.
- **id_event** - (Obrigatório para um novo grupo, Inteiro) Identifica o evento a que faz parte este grupo.
- **short_name** - (Obrigatório para um novo grupo, Texto 50 carateres) Abreviatura do grupo.
- **name** - (Obrigatório para um novo grupo, Texto 100 caracteres) Nome do grupo.

Response

```
{    "status": 200,    "result": "sucess"}
```

Método que retorna todos os grupos disponíveis para o evento.

```
auth1auth2http[s]://rpublish/api/groupings/(ID_EVENT)method : GET
```

Request

- **ID_EVENT** - Identificação do evento.

Response

```
{    "status": 200,    "result": [        {            ...        }    ]}
```

Categories (Categorias)

Método que permite adicionar ou alterar um **categories**.

```
auth1auth2http[s]://rpublish/api/categoriesmethod : POST
```

Request

Para tal deve ser enviado um array do tipo json, onde cada elemento do array deve ter os seguintes campos:

- **id_category** - (Obrigatório, Inteiro maior que 0) Identifica a categoria de uma forma única.
- **modality** - (Obrigatório para um novo categoria, Texto 50 caracteres) Abreviatura da modalidade de acordo com a tabela já referida no evento.

- **short_name** - (Obrigatório para um novo categoria, Texto 50 caracteres) Abreviatura da categoria.
- **name** - (Obrigatório para um novo categoria, Texto 255 caracteres) Nome da categoria.

Response

```
{    "status": 200,    "result": "sucess"}
```

Método que retorna uma categoria.

```
auth1auth2http[s]://rpublish/api/categories/(ID_CATEGORY)method : GET
```

Request

Response

```
{    "status": 200,    "result": {        "short_name": "C1",        "id_category": "1",        "name": "Class1",        "id_mod
```

Stages (Especiais)

Método que permite adicionar ou alterar um **stage(s)**.

```
auth1auth2http[s]://rpublish/api/stagesmethod : POST
```

Request

Para tal deve ser enviado um array do tipo json, onde cada elemento do array deve ter os seguintes campos os seguintes campos:

- **id_stage** - (Obrigatório para editar quando não é identificado o gpd_cod_h, Inteiro maior que 0) Identifica a especial de uma forma única.
- **id_event** - (Obrigatório para um novo stage, Inteiro maior que 0) Identifica o evento a que esta especial faz parte.
- **status** - (Obrigatório para um novo stage, [performing|ongoing|completed|canceled]) Estado da especial.
- **url_friendly** - (Obrigatório para um novo stage, Texto 100 caracteres) url de acesso rápido à especial.
- **short_name** - (Obrigatório para um novo stage, Texto 50 caracteres) Abreviatura da especial.
- **name** - (Obrigatório para um novo stage, Texto 100 caracteres) Nome da especial.
- **description** - (Obrigatório para um novo stage, Texto) Descrição da especial.
- **length** - (Obrigatório para um novo stage, decimal separado por um ponto) Cumprimento da especial.
- **start_date** - (Obrigatório para um novo stage, Data (YYYY-MM-DD HH-MM) *ISO 8601*) Data de inicio do evento.
- ***gpd_cod_h*** - *DEPRECATED* (Identifica a especial quando criada pelo GPD, Inteiro maior que 0) Identifica a especial de uma forma única no software externo.
- ***gpd_cod_esp*** - *DEPRECATED* (Inteiro maior que 0) Necessário para software externo.
- ***gpd_passagem*** - *DEPRECATED* (Inteiro maior que 0) Necessário para software externo.
- **splits** - Estrutura de intermédios do stage A estrutura de um intermédio trata-se de um array de splits composto cada um com os seguintes campos.
    - **id_split** - (Obrigatório, Inteiro maior que 0) Identifica o intermédio de uma forma única.
    - **sort_name** - (Obrigatório para um novo split, Texto 50 caracteres) Abreviatura do nome do spliter.
    - **name** - (Obrigatório para um novo split, Texto 255 caracteres) Nome do intermédio.
    - **length** - (Obrigatório para um novo split, decimal separado por um ponto) Cumprimento do intermédio.
    - **start_date** - (Obrigatório para um novo stage, Data (YYYY-MM-DD HH:MM) *ISO 8601*) Data de inicio do split.
    - ***gpd_cod_esp*** - *DEPRECATED* (Inteiro maior que 0) Necessário para software externo.
    - ***gpd_passagem*** - *DEPRECATED* (Inteiro maior que 0) Necessário para software externo.
    - ***gpd_ei*** - *DEPRECATED* (Inteiro maior que 0) Necessário para software externo.
- **grouping** - Array de ids de groupings.

Para manter compatibilidade com a aplicação externa GPD foi necessário adicionar três campos que permitam mapear os dados para o novo sistema. Estes campos foram adicionados temporariamente até à migração desta aplicação por uma versão na cloud.

Response

```
{    "status": 200,    "result": "sucess"}
```

Método que retorna as especiais de um evento.

```
auth1auth2http[s]://rpublish/api/stages/(ID_EVENT)method : GET
```

Request

Response

```
{    "status": 200,    "result": [        {            "status": "performing",            "length": "10.240",            "descri
```

Método que retorna os intermédios de uma especial.

```
auth1auth2http[s]://rpublish/api/splits/(ID_STAGE)method : GET
```

Request

Response

```
{    "status": 200,    "result": [        {            "length": "21.333",            "sort_name": "S2",            "start_date"
```

Competitors (Comcorrentes)

Método que permite adicionar ou alterar um **competitor(s)**.

```
auth1auth2http[s]://rpublish/api/competitorsmethod : POST
```

Request

Deve ser enviado um array do tipo json, onde cada elemento do array deve ter os seguintes campos:

- **id_competitor** - (Obrigatório para editar, Inteiro maior que 0) Identifica o concorrente de uma forma única.
- **id_event** - (Obrigatório para um novo competitor, Inteiro maior que 0) Identifica o evento a que esta especial faz parte.
- **status** - (Obrigatório para um novo competitor, [inscribed, participant]) Estado do concorrente.
- **pos** - (Obrigatório para um novo competitor, Inteiro) A posição do concorrente na grelha.
- **short_name** - (Texto 50 caracteres) Abreviatura da especial.
- **number** - (Obrigatório para um novo competitor, texto) Número que identifica o concorrente para o público (dorcal).
- **name1** - (Obrigatório para um novo competitor, Texto 255 carateres) primeiro nome do concorrente normalmente é o nome da equipa.
- **license1** - (Texto 50 carateres) licença do primeiro nome.
- **nac1** - (Texto 10 carateres) Nacionalidade do primeiro nome.
- **name2** - (Texto 255 carateres) segundo nome do concorrente normalmente é o nome da equipa.
- **license2** - (Texto 50 carateres) licença do segundo nome.
- **nac2** - (Texto 10 carateres) Nacionalidade do segundo nome.
- **name3** - (Texto 255 carateres) terceiro nome do concorrente normalmente é o nome da equipa.
- **license3** - (Texto 50 carateres) licença do terceiro nome.
- **nac3** - (Texto 10 carateres) Nacionalidade do terceiro nome.
- **mark** - (Texto 100 carateres) Marca quando necessária.
- **model** - (Texto 100 carateres) Modelo quando necessário.
- **categories** - Array com o short_name da categoria a que pertence.

Request

Response

```
{    "status": 200,    "result": {        "status": 200,        "result": "sucess"    }}
```

Método que retorna os concorrentes incritos/participantes de um evento.

```
auth1auth2http[s]://rpublish/api/competitors/(ID_EVENTO)method : GET
```

Request

Response

```
{    "status": 200,    "result": [            {            nac3: "ES",            nac2: "PT",            nac1: "
```

results (Provas)

Método que permite adicionar ou alterar um **resultados de provas(s)**.

```
auth1auth2http[s]://rpublish/api/resultsmethod : POST
```

Request

Deve ser enviado um objeto json com os seguintes campos:

- **type** - (Obrigatório para um novo resultado, [STAGE, OVERALL, SPLIP]) Qual o tipo do resultado
- **id_stage** - (Inteiro maior que 0) Identifica o stage a que este resultado corresponde.
- **id_split** - (Inteiro maior que 0) Identifica o split a que este resultado corresponde.
- **id_grouping** - (Obrigatório para um novo resultado, inteiro maior que 0) o grupo a que corresponde este resultado.
- ***gpd_cod_h*** - *DEPRECATED* (Identifica a especial quando criada pelo GPD, Inteiro maior que 0) Identifica a especial de uma forma única no software externo.
- ***gpd_ei*** - *DEPRECATED* (Inteiro maior que 0) Identifica a intermédio de uma forma única no software externo.
- **results** - um Array json onde cada elemento é composto por um objeto com os seguintes campos
    - **pos** - (Obrigatório, Inteiro) A posição do concorrente na grelha.
    - **id_competitor** - (Obrigatório, Inteiro) Identificação do concorrente.
    - **id_category** - (Obrigatório, Inteiro) Identificação da categoria do concorrente.
    - **time** - (Obrigatório, decimal) tempo.
    - **total_extra_time** - (Obrigatório, decimal) tempo de penalização.
    - **total_time** - (Obrigatório, decimal) tempo total.

Request

Response

```
{    "status": 200,    "result": {        "status": 200,        "result": "sucess"    }}
```

Método que retorna os resultados de uma especial.

```
auth1auth2http[s]://rpublish/api/results/(ID_EVENTO)-(ID_GROUPING)-(ID_CATEGORY)-(ID_STAGE)method : GET
```

Request

Response

```
{    "status": 200,    "result": ....}
```

Anexo 6

```
upstream publive {
    server 127.0.0.1:9001;
    server 127.0.0.1:9002;
    server 127.0.0.1:9003;
    server 127.0.0.1:9004;
}

upstream wspublive {
    server 127.0.0.1:9501;
}

server {
        listen   80;
        server_name wwr.stm.pt;

        access_log  /public_web/logs/stm/stm.pt/live.stm_access.log main;
        error_log   /public_web/logs/stm/stm.pt/live.stm_error.log;

        location ^~ /s/ {
                root /public_web/www/stm/live.realCrono/publish/htdocs;
                if ($query_string) {
                        expires max;
                }
        }

        error_page 400 401 402 403 404 405 406 407 408 500 502 503 504 /er
        location = /error.html {
              root /public_web/generic;
        }

        location = /favicon.ico {
                rewrite (.*) /s/favicon.ico;
        }

        location = /robots.txt {
                rewrite (.*) /s/robots.txt;
        }

        location /wsocket {
                proxy_pass_header Server;
                proxy_set_header Host $http_host;
                #proxy_set_header Host $host;
                proxy_redirect off;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Scheme $scheme;
                proxy_pass http://wspublive;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
```

```
                proxy_set_header Connection "upgrade";
                proxy_read_timeout 600s;
        }

        location / {
                proxy_pass_header Server;
                proxy_set_header Host $http_host;
                proxy_redirect off;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Scheme $scheme;
                proxy_pass http://publive;
        }
}
```

Anexo 7

```
#!/bin/bash
NEW='http://wwr.stm.pt/baiao2013/entries'
CORE=4
OLD='http://cam.stm.pt/falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=pt'
for NUM in 1 2 4 8 16 32 64 128 256 512 1024
do
    echo "$NUM request at same time"
    ab -n $1 -c $NUM -r -d $NEW >> new_$CORE.log
    sleep 5
    ab -n $1 -c $NUM -r -d $OLD >> old.log
    sleep 5
done
echo 'Test finish'
exit 0
```

Anexo 8

```
========================= ANTIGA APLICAÇÂO ========================
```

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      1
Time taken for tests:   1328.329 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      134355000 bytes
HTML transferred:       133780000 bytes
Requests per second:    3.76 [#/sec] (mean)
Time per request:       265.666 [ms] (mean)
Time per request:       265.666 [ms] (mean, across all concurrent requests
Transfer rate:          98.78 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       54   64  67.0     60    1460
Processing:   181  202  37.8    197    1075
Waiting:       68   79  17.9     77     958
Total:        238  266  78.8    257    1827
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

```

```
Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      2
Time taken for tests:   653.385 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      134355000 bytes
HTML transferred:       133780000 bytes
Requests per second:    7.65 [#/sec] (mean)
Time per request:       261.354 [ms] (mean)
Time per request:       130.677 [ms] (mean, across all concurrent requests
Transfer rate:          200.81 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       54   60  31.1     59    1457
Processing:   181  201  29.8    197     800
Waiting:       68   80  15.0     78     676
Total:        237  261  43.1    257    1659
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      4
Time taken for tests:   331.632 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      134355000 bytes
HTML transferred:       133780000 bytes
Requests per second:    15.08 [#/sec] (mean)
Time per request:       265.305 [ms] (mean)
Time per request:       66.326 [ms] (mean, across all concurrent requests)
Transfer rate:          395.64 [Kbytes/sec] received

Connection Times (ms)
```

```
              min  mean[+/-sd] median   max
Connect:       53   62  50.3     59    1596
Processing:   179  203  34.5    198    1018
Waiting:       67   81  13.2     78     357
Total:        235  265  62.4    258    1790
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      8
Time taken for tests:   169.267 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      134355000 bytes
HTML transferred:       133780000 bytes
Requests per second:    29.54 [#/sec] (mean)
Time per request:       270.827 [ms] (mean)
Time per request:       33.853 [ms] (mean, across all concurrent requests)
Transfer rate:          775.14 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       55   62  59.9     60    3057
Processing:   181  209  27.8    203     638
Waiting:       68   88  17.1     83     369
Total:        238  271  67.1    263    3258
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

```

```
Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      16
Time taken for tests:   91.468 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      134355000 bytes
HTML transferred:       133780000 bytes
Requests per second:    54.66 [#/sec] (mean)
Time per request:       292.698 [ms] (mean)
Time per request:       18.294 [ms] (mean, across all concurrent requests)
Transfer rate:          1434.45 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       55   61  29.5     60    1437
Processing:   179  232  41.3    221     692
Waiting:       68  111  35.9    102     365
Total:        237  292  50.5    282    1651
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      32
Time taken for tests:   59.311 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      134355000 bytes
HTML transferred:       133780000 bytes
Requests per second:    84.30 [#/sec] (mean)
Time per request:       379.589 [ms] (mean)
Time per request:       11.862 [ms] (mean, across all concurrent requests)
Transfer rate:          2212.18 [Kbytes/sec] received

Connection Times (ms)
```

```
              min  mean[+/-sd] median   max
Connect:       54   62  44.9     60    1494
Processing:   185  317  94.0    303    1183
Waiting:       70  193  81.2    183    1066
Total:        243  379 103.5    364    1751
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      64
Time taken for tests:   49.906 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      134355000 bytes
HTML transferred:       133780000 bytes
Requests per second:    100.19 [#/sec] (mean)
Time per request:       638.791 [ms] (mean)
Time per request:       9.981 [ms] (mean, across all concurrent requests)
Transfer rate:          2629.09 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       55   61  18.7     61    1366
Processing:   187  575 175.5    549    1308
Waiting:       72  443 170.9    419    1152
Total:        247  636 177.0    610    2227
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

```

```
Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      128
Time taken for tests:   48.479 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      134355000 bytes
HTML transferred:       133780000 bytes
Requests per second:    103.14 [#/sec] (mean)
Time per request:       1241.059 [ms] (mean)
Time per request:       9.696 [ms] (mean, across all concurrent requests)
Transfer rate:          2706.46 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       55   81 231.4     61    3072
Processing:   285 1151 358.8   1106    2818
Waiting:      121  930 359.2    874    2700
Total:        347 1232 413.5   1176    4062
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cam.stm.pt (be patient)


Server Software:        nginx
Server Hostname:        cam.stm.pt
Server Port:            80

Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=p
Document Length:        26756 bytes

Concurrency Level:      256
Time taken for tests:   32.892 seconds
Complete requests:      5000
Failed requests:        2039
   (Connect: 0, Receive: 0, Length: 2039, Exceptions: 0)
Write errors:           0
Non-2xx responses:      2039
Total transferred:      79989180 bytes
HTML transferred:       79375439 bytes
Requests per second:    152.01 [#/sec] (mean)
Time per request:       1684.056 [ms] (mean)
Time per request:       6.578 [ms] (mean, across all concurrent requests)
Transfer rate:          2374.90 [Kbytes/sec] received
```

```
289
290 Connection Times (ms)
291               min  mean[+/-sd] median   max
292 Connect:       54  156 660.3     61    9064
293 Processing:   302 1494 784.1   1378    6481
294 Waiting:      178 1270 617.5   1190    5619
295 Total:        362 1650 1045.4   1564   11499
296 This is ApacheBench, Version 2.3 <$Revision: 655654 $>
297 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
298 Licensed to The Apache Software Foundation, http://www.apache.org/
299
300 Benchmarking cam.stm.pt (be patient)
301
302
303 Server Software:        nginx
304 Server Hostname:        cam.stm.pt
305 Server Port:            80
306
307 Document Path:          /falperra2013/?pg=1&e=37&s=111&cat=310&h=1369&lg=
308 Document Length:        46 bytes
309
310 Concurrency Level:      512
311 Time taken for tests:   12.840 seconds
312 Complete requests:      5000
313 Failed requests:        4912
314    (Connect: 0, Receive: 0, Length: 4912, Exceptions: 0)
315 Write errors:           0
316 Non-2xx responses:      3701
317 Total transferred:      37187128 bytes
318 HTML transferred:       36430269 bytes
319 Requests per second:    389.41 [#/sec] (mean)
320 Time per request:       1314.802 [ms] (mean)
321 Time per request:       2.568 [ms] (mean, across all concurrent requests)
322 Transfer rate:          2828.34 [Kbytes/sec] received
323
324 Connection Times (ms)
325               min  mean[+/-sd] median   max
326 Connect:       54  398 1493.1     67    9112
327 Processing:    57  743 1054.4     79    4369
328 Waiting:       57  672 940.1     79    3657
329 Total:        115 1141 1825.3    163   12114
330 This is ApacheBench, Version 2.3 <$Revision: 655654 $>
331 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
332 Licensed to The Apache Software Foundation, http://www.apache.org/
333
334 Benchmarking cam.stm.pt (be patient)
335
336
```

```
337
338
339
340
341
342 ========================= NOVA APLICAÇÂO ========================
343
344
345
346
347
348
349
350 This is ApacheBench, Version 2.3 <$Revision: 655654 $>
351 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
352 Licensed to The Apache Software Foundation, http://www.apache.org/
353
354 Benchmarking wwr.stm.pt (be patient)
355
356
357 Server Software:        TornadoServer/3.1
358 Server Hostname:        wwr.stm.pt
359 Server Port:            80
360
361 Document Path:          /baiao2013/entries
362 Document Length:        21337 bytes
363
364 Concurrency Level:      1
365 Time taken for tests:   1401.696 seconds
366 Complete requests:      5000
367 Failed requests:        0
368 Write errors:           0
369 Total transferred:      108275000 bytes
370 HTML transferred:       106685000 bytes
371 Requests per second:    3.57 [#/sec] (mean)
372 Time per request:       280.339 [ms] (mean)
373 Time per request:       280.339 [ms] (mean, across all concurrent request
374 Transfer rate:          75.44 [Kbytes/sec] received
375
376 Connection Times (ms)
377               min  mean[+/-sd] median   max
378 Connect:       54   61  28.2     60    1259
379 Processing:   149  219  36.9    212     940
380 Waiting:       87  104  19.6     96     361
381 Total:        208  280  47.5    272    1487
382 This is ApacheBench, Version 2.3 <$Revision: 655654 $>
383 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
384 Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
385
386 Benchmarking wwr.stm.pt (be patient)
387
388
389 Server Software:        TornadoServer/3.1
390 Server Hostname:        wwr.stm.pt
391 Server Port:            80
392
393 Document Path:          /baiao2013/entries
394 Document Length:        21337 bytes
395
396 Concurrency Level:      2
397 Time taken for tests:   760.046 seconds
398 Complete requests:      5000
399 Failed requests:        0
400 Write errors:           0
401 Total transferred:      108275000 bytes
402 HTML transferred:       106685000 bytes
403 Requests per second:    6.58 [#/sec] (mean)
404 Time per request:       304.018 [ms] (mean)
405 Time per request:       152.009 [ms] (mean, across all concurrent request
406 Transfer rate:          139.12 [Kbytes/sec] received
407
408 Connection Times (ms)
409               min  mean[+/-sd] median   max
410 Connect:       54   65  66.6     60    1647
411 Processing:   152  239  56.1    225    1133
412 Waiting:       84  118  31.4    111     593
413 Total:        212  304  90.0    286    1976
414 This is ApacheBench, Version 2.3 <$Revision: 655654 $>
415 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
416 Licensed to The Apache Software Foundation, http://www.apache.org/
417
418 Benchmarking wwr.stm.pt (be patient)
419
420
421 Server Software:        TornadoServer/3.1
422 Server Hostname:        wwr.stm.pt
423 Server Port:            80
424
425 Document Path:          /baiao2013/entries
426 Document Length:        21337 bytes
427
428 Concurrency Level:      4
429 Time taken for tests:   407.588 seconds
430 Complete requests:      5000
431 Failed requests:        0
432 Write errors:           0
```

```
Total transferred:      108275000 bytes
HTML transferred:       106685000 bytes
Requests per second:    12.27 [#/sec] (mean)
Time per request:       326.071 [ms] (mean)
Time per request:       81.518 [ms] (mean, across all concurrent requests
Transfer rate:          259.42 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       54   61  39.5     60    1456
Processing:   153  264  55.8    255     908
Waiting:       86  148  50.1    141     402
Total:        210  326  68.8    315    1664
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking wwr.stm.pt (be patient)


Server Software:        TornadoServer/3.1
Server Hostname:        wwr.stm.pt
Server Port:            80

Document Path:          /baiao2013/entries
Document Length:        21337 bytes

Concurrency Level:      8
Time taken for tests:   253.838 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      108275000 bytes
HTML transferred:       106685000 bytes
Requests per second:    19.70 [#/sec] (mean)
Time per request:       406.141 [ms] (mean)
Time per request:       50.768 [ms] (mean, across all concurrent requests
Transfer rate:          416.55 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       54   61  37.3     60    1395
Processing:   158  345  90.9    339    1281
Waiting:       87  228  87.4    223    1164
Total:        220  406  97.9    399    1666
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
481
482 Benchmarking wwr.stm.pt (be patient)
483
484
485 Server Software:        TornadoServer/3.1
486 Server Hostname:        wwr.stm.pt
487 Server Port:            80
488
489 Document Path:          /baiao2013/entries
490 Document Length:        21337 bytes
491
492 Concurrency Level:      16
493 Time taken for tests:   171.840 seconds
494 Complete requests:      5000
495 Failed requests:        0
496 Write errors:           0
497 Total transferred:      108275000 bytes
498 HTML transferred:       106685000 bytes
499 Requests per second:    29.10 [#/sec] (mean)
500 Time per request:       549.887 [ms] (mean)
501 Time per request:       34.368 [ms] (mean, across all concurrent requests
502 Transfer rate:          615.33 [Kbytes/sec] received
503
504 Connection Times (ms)
505               min  mean[+/-sd] median   max
506 Connect:       54   62  53.7     60    1583
507 Processing:   293  487 121.0    466    1197
508 Waiting:      190  371 119.3    351    1082
509 Total:        353  549 132.1    526    2084
510 This is ApacheBench, Version 2.3 <$Revision: 655654 $>
511 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
512 Licensed to The Apache Software Foundation, http://www.apache.org/
513
514 Benchmarking wwr.stm.pt (be patient)
515
516
517 Server Software:        TornadoServer/3.1
518 Server Hostname:        wwr.stm.pt
519 Server Port:            80
520
521 Document Path:          /baiao2013/entries
522 Document Length:        21337 bytes
523
524 Concurrency Level:      32
525 Time taken for tests:   170.021 seconds
526 Complete requests:      5000
527 Failed requests:        0
528 Write errors:           0
```

```
Total transferred:      108275000 bytes
HTML transferred:       106685000 bytes
Requests per second:    29.41 [#/sec] (mean)
Time per request:       1088.136 [ms] (mean)
Time per request:       34.004 [ms] (mean, across all concurrent requests
Transfer rate:          621.91 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       54   61  36.2     60    1443
Processing:   345 1024 311.1    980    2357
Waiting:      230  908 310.8    863    2244
Total:        402 1085 312.6   1041    2596
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking wwr.stm.pt (be patient)


Server Software:        TornadoServer/3.1
Server Hostname:        wwr.stm.pt
Server Port:            80

Document Path:          /baiao2013/entries
Document Length:        21337 bytes

Concurrency Level:      64
Time taken for tests:   165.688 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      108275000 bytes
HTML transferred:       106685000 bytes
Requests per second:    30.18 [#/sec] (mean)
Time per request:       2120.811 [ms] (mean)
Time per request:       33.138 [ms] (mean, across all concurrent requests
Transfer rate:          638.17 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       54   60  17.1     60    1255
Processing:   532 2048 448.4   2091    3160
Waiting:      424 1932 448.2   1974    3047
Total:        590 2109 448.5   2151    3222
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
577
578 Benchmarking wwr.stm.pt (be patient)
579
580
581 Server Software:        TornadoServer/3.1
582 Server Hostname:        wwr.stm.pt
583 Server Port:            80
584
585 Document Path:          /baiao2013/entries
586 Document Length:        21337 bytes
587
588 Concurrency Level:      128
589 Time taken for tests:   170.480 seconds
590 Complete requests:      5000
591 Failed requests:        0
592 Write errors:           0
593 Total transferred:      108275000 bytes
594 HTML transferred:       106685000 bytes
595 Requests per second:    29.33 [#/sec] (mean)
596 Time per request:       4364.300 [ms] (mean)
597 Time per request:       34.096 [ms] (mean, across all concurrent requests
598 Transfer rate:          620.23 [Kbytes/sec] received
599
600 Connection Times (ms)
601               min  mean[+/-sd] median   max
602 Connect:       54   80 240.0     60    4655
603 Processing:   545 4233 489.5   4244    5662
604 Waiting:      432 4118 489.9   4128    5541
605 Total:        610 4313 524.5   4311    8913
606 This is ApacheBench, Version 2.3 <$Revision: 655654 $>
607 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
608 Licensed to The Apache Software Foundation, http://www.apache.org/
609
610 Benchmarking wwr.stm.pt (be patient)
611
612
613 Server Software:        TornadoServer/3.1
614 Server Hostname:        wwr.stm.pt
615 Server Port:            80
616
617 Document Path:          /baiao2013/entries
618 Document Length:        21337 bytes
619
620 Concurrency Level:      256
621 Time taken for tests:   165.430 seconds
622 Complete requests:      5000
623 Failed requests:        0
624 Write errors:           0
```

```
Total transferred:      108275000 bytes
HTML transferred:       106685000 bytes
Requests per second:    30.22 [#/sec] (mean)
Time per request:       8470.030 [ms] (mean)
Time per request:       33.086 [ms] (mean, across all concurrent requests
Transfer rate:          639.17 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       53  159 691.8     60    9071
Processing:   408 8102 1182.4   8346    9578
Waiting:      298 7986 1182.4   8230    9452
Total:        473 8260 1265.1   8420   17632
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking wwr.stm.pt (be patient)


Server Software:        TornadoServer/3.1
Server Hostname:        wwr.stm.pt
Server Port:            80

Document Path:          /baiao2013/entries
Document Length:        21337 bytes

Concurrency Level:      512
Time taken for tests:   174.925 seconds
Complete requests:      5000
Failed requests:        47
   (Connect: 0, Receive: 12, Length: 23, Exceptions: 12)
Write errors:           0
Non-2xx responses:      11
Total transferred:      107796614 bytes
HTML transferred:       106211587 bytes
Requests per second:    28.58 [#/sec] (mean)
Time per request:       17912.362 [ms] (mean)
Time per request:       34.985 [ms] (mean, across all concurrent requests
Transfer rate:          601.80 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  388 1471.6     60    9095
Processing:    57 16616 3497.3  17699   20998
Waiting:        0 16450 3582.1  17581   19014
Total:        117 17004 3550.4  17801   27424
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
```

```
673 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
674 Licensed to The Apache Software Foundation, http://www.apache.org/
675
676 Benchmarking wwr.stm.pt (be patient)
677 This is ApacheBench, Version 2.3 <$Revision: 655654 $>
678 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
679 Licensed to The Apache Software Foundation, http://www.apache.org/
680
681
682
683
684
```