Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Fábio Daniel Martinho Ferreira

Otimização de desempenho de uma aplicação de participação pública

Outubro 2015

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Fábio Daniel Martinho Ferreira

Otimização de desempenho de uma aplicação de participação pública

Tese de Mestrado

Sistemas e Tecnologias de Informação para as Organizações

Professor Doutor Francisco Ferreira Francisco

Outubro 2015

RESUMO

A massificação do acesso à Internet, juntamente com a evolução tecnológica, proporcionaram a criação de aplicações web cada vez mais exigentes e complexas. Entre os requisitos para o sucesso de um *website*, o desempenho tem um papel fundamental.

Portanto, este trabalho de investigação e desenvolvimento procura efetuar um levantamento das principais técnicas de melhoria do desempenho de aplicações web para aplica-las num contexto de uma aplicação complexa.

Uma das principais motivações para a realização deste trabalho de investigação e desenvolvimento está relacionada com a necessidade de melhorar a performance de uma plataforma web de participação pública, designada Liberopinion. Esta plataforma tem vindo a ser alvo de um número crescente de acessos (em simultâneo) e de um acréscimo da implementação de novas e complexas funcionalidades, o que tem provocado alguma degradação no desempenho. Neste trabalho apresentam-se também as técnicas que tiveram mais impacto para a melhoria de desempenho da plataforma Liberopinion.

Nesta dissertação apresentam-se e analisam-se os resultados obtidos a partir das medições do desempenho, antes e após a implementação das diferentes técnicas de otimização nas diversas instâncias da plataforma Liberopinion em produção. Entre os principais resultados, a combinação de diferentes técnicas de otimização do desempenho destaca-se como a abordagem com melhorias mais significativas na performance de aplicações Web.

Resumindo, acredita-se fortemente que o desempenho das páginas de um *website* tem impacto na taxa de rejeição. Quanto mais tempo uma página demorar a ser carregada, maior é a probabilidade de o utilizador sair do *website* depois de visualizar a primeira página.

ABSTRACT

The mass access to the Internet, along with technological developments, led to the creation of increasingly demanding and complex web applications. Among the requirements for the success of a website, performance plays a key role.

Therefore, this work of research and development seeks to conduct a survey of the main techniques to improve the performance of web applications and apply them in the context of a complex application.

One of the main motivations for conducting this work of research and development comes from the need of performance improvements in a web platform for public participation, designated Liberopinion. This platform has experienced a growing number of accesses (simultaneously) and an increase of new and complex features implemented, which has caused degradation in performance. In this paper we present also the techniques leading to the highest improvements on the Liberopinion platform.

This thesis presents and analyzes the results obtained from performance measurements, before and after applying optimization techniques to different instances of the Liberopinion platform in production. Among the key findings, the combination of different performance optimization techniques stands out as the approach leading to the most significant improvements in the performance of Web applications.

In summary, it is strongly believed that webpage performance within a website has direct impact in the bounce rate. The longer it takes to load a page, the higher the probability that the user leaves the website after viewing the first page.

PALAVRAS CHAVE

Monitorização de desempenho

Aplicações Web

Técnicas de Otimização

KEY WORDS

Performance monitoring

Web Applications

Optimization techniques

AGRADECIMENTOS

Começo por agradecer ao meu orientador, Professor Francisco Ferreira Francisco, pelos conselhos, apoio e disponibilidade em todas as fases da elaboração do trabalho.

Agradeço aos cofundadores da Libertrium, Professor Artur Sousa e Eng.º Pedro Agante, por disponibilizarem os meios para colocar a plataforma de medição de desempenho em produção, deixarem utilizar o projeto Liberopinion para desenvolver o trabalho e, sobretudo, pelo seu companheirismo e sugestões.

Quero agradecer aos meus amigos por todo o incentivo, em particular, à Patrícia Almeida e Tiago Monteiro, pelo seu contributo mais direto no trabalho da dissertação.

Por fim, agradeço à minha família, em especial, aos meus pais, por todo o seu apoio e conselhos. O meu desenvolvimento académico não teria sido possível sem o seu esforço e dedicação.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ABREVIATURAS E SIGLAS

Lista de abreviaturas e siglas (ordenadas por ordem alfabética)

API	Application Programming Interface
AJAX	Asynchronous Javascript and XML
CDN	Content Delivery Network
CSS	Cascading Style Sheets
GIF	Graphics Interchange Format
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
PDF	Portable Document Format
PNG	Portable Network Graphics
SPA	Single Page Application
TTL	Time-to-Live
URI	Uniform Resource Identifier
URL	Uniform Resource Location
XML	eXtensible Markup Language

1 Introdução

A massificação do acesso à Internet, a evolução tecnológica e a necessidade de serviços em tempo real tornaram a Web num ambiente propício para a utilização de aplicações. Qualquer *smartphone*, *tablet*, portátil e computador tem um *browser* instalado que, com ligação à Internet, torna as aplicações Web acessíveis em qualquer momento e lugar.

Inicialmente, os *websites* eram essencialmente constituídos por páginas estáticas, ligadas através de hiperligações, mas as recentes inovações tecnológicas permitiram que eles se tornassem mais funcionais e suportassem mais requisitos. As aplicações Web atuais adaptam-se a ambientes de negócio e de serviços, tornando-se, assim, mais complexas. Apesar de os operadores disponibilizarem cada vez mais melhores condições acesso à rede, por vezes as aplicações baseadas na Internet estão a ficar mais lentas [1]. De acordo com informação revelada pelo HTTP Archive (http://httparchive.org), um *website*, atualmente (2015), ocupa em média 2.1 MegaBytes, o dobro de em 2012. Uma maior utilização de conteúdos multimédia, como fotografias, vídeos e música, contribui para que os tempos de resposta das aplicações se tornem cada vez mais longos.

1.1 Enquadramento e Motivação

A complexidade e o ambiente competitivo e social da Web tornam o desempenho um requisito fundamental para o sucesso das aplicações Web. A performance está diretamente ligada à experiência de utilização, pois os tempos de resposta no acesso afetam o comportamento do ser humano [2]. Por outro lado, a otimização dos tempos de resposta melhora o posicionamento de uma aplicação Web nos rankings dos motores de busca [3], podendo, assim, contribuir para aumentar o seu número de visitantes e, consequentemente o seu sucesso.

A necessidade de otimizar a performance de uma aplicação Web, designada Liberopinion (http://liberopinion.com) e cujo objetivo é a mediação digital para a participação pública,

despertou o interesse em procurar informação sobre técnicas de otimização que resolvessem ou atenuassem os problemas de desempenho dessa aplicação, tornando-se, assim, na principal motivação para a realização deste trabalho de investigação e desenvolvimento.

As aplicações destinadas à participação pública, devido às suas características eminentemente inovadoras, carecem constantemente de novas funcionalidades, o que aumenta o nível de complexidade do seu desenvolvimento e manutenção. A complexidade, como referido anteriormente, pode ter um impacto negativo nos tempos de acesso, uma vez que, entre outros motivos, é preciso transferir ficheiros com as novas funcionalidades que vão sendo adicionadas. Foi precisamente o que aconteceu com a aplicação Liberopinion. As constantes atualizações com novas funcionalidades, cada vez mais exigentes e complexas, e o aumento do número de utilizadores, provocaram uma degradação considerável nos tempos de acesso a algumas das suas páginas. Em alguns casos, provocou mesmo o não carregamento da página no *backoffice*, por ser atingido o valor limite de tempo de espera (*time out*). Portanto, caso não fossem tomadas medidas conducentes à melhoria da performance desta aplicação, estaria em risco o seu uso com parâmetros aceitáveis para a experiência de utilização.

1.2 Objetivos

Com este trabalho de investigação e desenvolvimento pretende-se efetuar um levantamento das principais técnicas de melhoria do desempenho de aplicações web, para depois as aplicar, em contexto real (na aplicação de participação pública Liberopinion, anteriormente referida), juntamente com algumas combinações dessas técnicas. Para o efeito, é necessário investigar os processos envolvidos no acesso às páginas Web, de modo a compreender melhor as métricas de performance em estudo.

A medição é crucial para avaliar o impacto das técnicas de otimização implementadas nos valores das métricas de performance e no comportamento dos utilizadores. Deste modo, neste trabalho procura-se também analisar alguns métodos de recolha e análise de dados de desempenho e apresentar algumas sugestões de medição. É ainda objetivo deste projeto apresentar as técnicas que tiveram mais impacto para a melhoria de desempenho da aplicação Liberopinion.

Em suma, tendo em conta o problema subjacente, este projeto tem uma componente de investigação e outra de desenvolvimento. Assim, pretendem-se cumprir os seguintes objetivos.

- Investigação
    - Identificar os principais processos e fatores que contribuem para a degradação da performance.
    - Identificar métricas a partir das quais se medirá o desempenho das aplicações Web.
    - Procurar ferramentas que permitam a recolha e a análise de dados sobre a performance.
    - Apresentar as técnicas que tenham mais impacto para a melhoria de desempenho da aplicação Liberopinion.
- Desenvolvimento

- Criar uma solução de monitorização e controlo do desempenho de aplicações Web.
- Melhorar o tempo de acesso a uma aplicação Web, implementando técnicas de melhoria de performance a um caso real.
- Analisar e tirar conclusões do impacto das melhorias na utilização da plataforma.

1.3 Metodologia e Procedimento

Face à natureza do projeto e às metodologias estudadas no decorrer do curso, conclui-se que a metodologia mais adequada para atingir os objetivos, é a investigação-acção. De acordo com Koshy [4], o principal papel da investigação-ação é facilitar o estudo de aspetos de índole prática, seja no contexto da introdução de uma ideia inovadora ou avaliação e reflexão da eficácia de uma prática corrente, com o objetivo de melhorar esses aspetos.

Dos vários modelos existentes que aplicam a metodologia investigação-ação, todos envolvem uma espiral de ciclos de autorreflexão. Cada ciclo é composto por um conjunto de fases. Porém o modelo que se adapta melhor ao problema em estudo é o proposto por Kemmis e McTaggart [5]. Os autores argumentam que a investigação-ação envolve uma espiral de ciclos de autorreflexão:

- Planificar a mudança,
- Agir e observar o processo e as consequências da mudança,
- Refletir sobre esses processos e consequências e, em seguida, voltar a planificar,
- Agir e observar,
- Refletir,
- E assim sucessivamente...

A Figura 1 ilustra o modelo proposto por Kemmis e McTaggart.

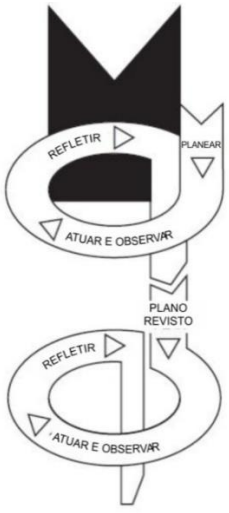


Figura 1: Modelo de investigação-ação proposto por Kemmis e McTaggart [5]

No entanto, é importante salientar que os autores aconselham a não utilizar esta estrutura de forma rígida. Eles afirmam que, na realidade, a espiral de ciclos de planeamento, ação e observação e de reflexão não deve ser seguida rigorosamente. As fases, segundo eles, sobrepõem-se, e os planos iniciais rapidamente se tornam obsoletos, em função da aprendizagem proporcionada através da experiência. Na realidade, o processo é suscetível de ser mais fluido e aberto.

A seguir apresenta-se uma breve descrição do procedimento adotado no decurso deste trabalho de investigação-ação. Assim, como referido anteriormente, após algumas atualizações da plataforma Liberopinion com funcionalidades complexas e um aumento considerável de número de visitas dos utilizadores, o *feedback* dado pelos administradores de algumas instâncias da plataforma apontava para alguma insatisfação sobre o tempo de acesso a algumas páginas.

Assim, numa fase preparatória do trabalho, procedeu-se à formulação do problema (degradação do desempenho da aplicação Liberopinion) e à identificação dos objetivos em estudo (melhoria do desempenho da aplicação Liberopinion).

Depois, na primeira fase do ciclo de investigação-ação, no planeamento, procedeu-se à elaboração do plano: efetuar uma revisão da literatura, essencialmente com o intuito de identificar as principais métricas de medida do desempenho e de identificar as principais técnicas de otimização do desempenho de aplicações Web.

Na fase 2 do ciclo de investigação-ação, na ação e observação, desenvolveu-se uma aplicação para medir as métricas de avaliação do desempenho de aplicações Web e implementaram-se algumas técnicas de otimização de desempenho nas instâncias da plataforma Liberopinion em produção. Depois, utilizou-se a aplicação desenvolvida para a avaliação do desempenho, juntamente com o Google Analytics, para medir as métricas de desempenho e de utilização da plataforma. Ainda na fase 2, observou-se o comportamento das várias instâncias da plataforma e analisaram-se os dados resultantes das medições de desempenho. Finalmente, na terceira fase, procedeu-se a uma reflexão sobre os resultados da medição e da observação do comportamento da plataforma. Uma vez que o *feedback* recebido, nessa altura, por parte dos administradores que tinham inicialmente manifestado alguma insatisfação sobre o tempo de acesso a algumas páginas da plataforma, ser já positivo, mostrando total satisfação com o desempenho atual da plataforma Liberopinion, decidiu-se não implementar um novo ciclo de investigação-ação.

Todavia, identificaram-se ainda alguns aspetos que requeriam melhoria, que são apresentados na conclusão desta dissertação, como trabalho futuro. Espera-se que este projeto de investigação e desenvolvimento possa continuar a evoluir.

Para o desenvolvimento da plataforma Liberopinion decidiu-se adotar uma metodologia que considerasse o manifesto ágil [6], isto é, que privilegiasse o desenvolvimento rápido e contínuo de *software*, gerido em períodos curtos, que aceitasse alterações de requisitos, mesmo numa fase tardia do ciclo de desenvolvimento (que abraçasse e potenciasse a mudança), e que promovesse a colaboração [7].

1.4 Estrutura da dissertação

Nesta secção descreve-se a estrutura da dissertação. Assim, neste primeiro capítulo, começa-se por contextualizar o estudo. Depois, formula-se o problema e apresentam-se as principais motivações do trabalho. Expõem-se os objetivos operacionais que traduzem as motivações e que orientaram as atividades deste projeto de investigação e desenvolvimento. Finalmente, a descrição da organização da dissertação, aqui a ser realizada, encerra este primeiro capítulo.

No capítulo 2, descreve-se o comportamento humano face aos tempos de espera que está sujeito enquanto navega na Web, a importância da performance no ciclo de vida de uma aplicação Web, com base em estudos estatísticos, e os processos que ocorrem na navegação, que condicionam o tempo de acesso às páginas Web.

No capítulo 3, enumeram-se e descrevem-se as várias métricas de performance, através das quais podem ser avaliadas as aplicações Web, bem como os métodos de recolha de dados.

No capítulo 4, são apresentadas algumas ferramentas existentes para analisar a performance de uma aplicação Web.

No capítulo 5, é descrito um conjunto de boas práticas para melhorar os valores das métricas de performance. São também mostrados resultados empíricos sobre os ganhos de algumas delas.

No capítulo 6, é apresentada a plataforma Liberopinion, cuja necessidade de melhoria de desempenho serviu como principal motivação para a realização deste trabalho. Alias, as várias instâncias desta plataforma foram utilizadas ao longo do trabalho para testar as técnicas de melhoria de desempenho e para efetuar a recolha de dados resultante da medida do desempenho.

O capítulo 7 aborda uma aplicação desenvolvida especificamente para monitorizar e analisar os dados de desempenho e para auxiliar na implementação de algumas técnicas de otimização.

No capítulo 8 são divulgados os dados recolhidos a partir das ações de monitorização adotadas, bem como os processos e resultados das melhorias (de desempenho) implementadas.

Por fim, no capítulo 9, são apresentadas as principais conclusões deste trabalho de investigação e desenvolvimento, revêm-se os principais objetivos traçados e apresentam-se algumas propostas de trabalho a ser desenvolvido futuramente.

2 Performance na Web

A Internet faz parte do quotidiano. Utiliza-se para trabalhar, entreter e informar sobre os mais diversos temas. O constante acesso e a diversidade de conteúdo aí presente, torna os seus utilizadores exigentes no que toca ao tempo de espera, seja a ler um jornal *online* ou realizar uma compra *online*. Apresentam-se no presente capítulo o impacto da performance na web e as conclusões retiradas de estudos estatísticos com intuito de demonstrar a importância do bom do desempenho nos *websites*.

2.1 Perceção Humana

Vários estudos feitos nos últimos 50 anos revelam que a perceção humana mantém-se constante quando submetida a diferentes tempos de resposta. Os comportamentos humanos face a três limites de tempos [8] são os seguintes:

- **0.1 segundos** é o tempo limite para que o utilizador sinta que o sistema reage instantaneamente às suas ações. Não há necessidade de fornecer feedback acerca do progresso da ação, apenas o seu resultado.
- **1 segundo** é o tempo limite para que a concentração do utilizador não seja interrompida, embora detete o atraso. Não é necessário dar feedback, mas perde-se a sensação que se age diretamente sobre o sistema.
- **10 segundos** é o tempo limite para manter a atenção do utilizador. Para tempos de espera superiores, o utilizador sente a necessidade de realizar outras tarefas. Deve-se fornecer feedback sobre o progresso da ação que está a ser executada. O feedback do progresso

da ação é essencial para o utilizador ter noção que não ocorreu nenhum problema para que não desista de esperar até a ação ser concluída.

O tempo é uma grandeza física medida quantitativamente com instrumentos, mas quantas vezes existem situações em que diferentes pessoas têm perceção diferente acerca da duração de determinado evento. Os seres humanos são diferentes e estão sujeitos a distintos estados emocionais, provocando diversas reações a diferentes tempos de resposta. Quanto menor for o seu tempo disponível, maior será a ansiedade e necessidade de aceder rapidamente a uma página web. O psicólogo Mihály Csíkszentmihályi formulou a teoria do *Flow* [2], muito utilizada no desenvolvimento dos jogos mas que pode ser aplicada em outros contextos, como por exemplo, as interfaces Web. O *Flow* é o estado mental de alguém que atinge a concentração numa atividade de tal forma, que se encontra imerso nessa experiência perdendo a noção espacial e temporal. Segundo a teoria, a realização de uma atividade influencia os sentimentos do utilizador, que são regulados pela dificuldade da tarefa e a perícia do utilizador. A Figura 2 ilustra os sentimentos de acordo a dificuldade sentida e a perícia do utilizador.

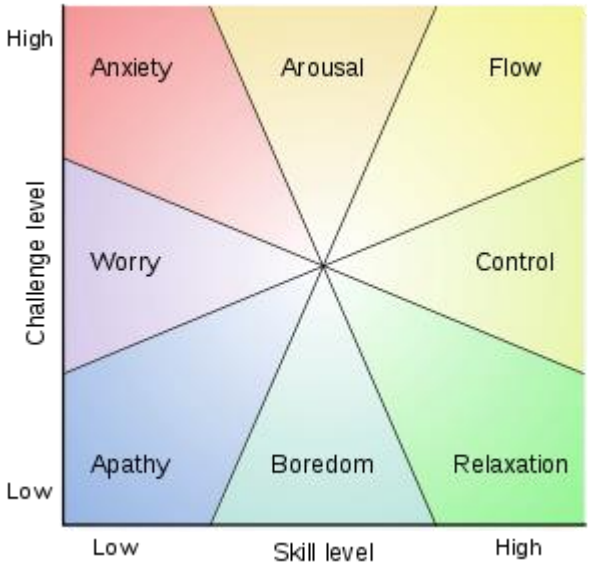


Figura 2: Estados emocionais na prática de uma atividade

Elevados tempos de espera tornam a execução da tarefa mais difícil, provocando estados emocionais como a ansiedade e a preocupação que degradam a experiência de utilização. A navegação Web deve ser uma boa experiência, uma tarefa que o utilizador deve usufruir e que não o frustre. Sem dúvida que o tempo de resposta de uma página Web interrompe a concentração da execução de uma tarefa, sendo um aspeto a ter em conta na construção das aplicações Web.

O tempo de carregamento da página afeta a utilização da web, tornando a sua experiência mais frustrante ou fluída. A seguir apresentam-se dados empíricos que sustentam a importância da performance.

2.2 Estudos estatísticos

Várias empresas têm a preocupação de realizar estudos sobre o desempenho das suas aplicações. A análise de dados reais de desempenho recolhidos resulta depois, na publicação de conclusões interessantes acerca do comportamento dos utilizadores.

Um estudo realizado pela Radware [9], em 2014, revela informação sobre a performance de 100 dos mais populares vendedores *online*. Nas amostras analisadas, a média de início da renderização de uma página, ou seja quando surge algum elemento visual, é de 5,4 segundos. A média do tempo que a página demora a encontrar-se completa para ser utilizada é de 10,7 segundos. É referido pela maioria dos compradores *online* envolvidos no estudo, que eles abandonam um website quando este demora mais de 3 segundos a ficar disponível para interação.

Noutro estudo realizado pela Gomez, em 2010 [10], retiram-se conclusões semelhantes, os utilizadores têm a expectativa que uma página seja carregada em dois segundos, e depois de 3 segundos mais de 40% dos utilizadores abandonam-na. A taxa de rejeição é maior, quanto mais tempo se demorar a aceder às páginas Web. O mesmo estudo conclui ainda que 88% dos compradores estão menos recetivos a reutilizar o mesmo *website* para realizar as suas compras após uma má experiência de utilização. Esta ocorre quando não se consegue concluir alguma tarefa, é extremamente demorosa ou confusa de concluir.

Um estudo realizado pela empresa Akamai, refere que 75% dos utilizadores realiza compras no *website* da concorrência em vez de esperar [10], deixando de fazer compras onde tiveram uma má experiência de utilização. A má experiência de utilização tira confiança ao utilizador para utilizar uma aplicação Web, optando por fazer as suas tarefas na concorrência.

Segundo a Google [11], os utilizadores voltam aos *websites* com melhores tempos de carregamento. Quando sofrem algum atraso, existe uma quebra no número de pesquisas ao motor de busca. Os utilizadores submetidos a testes controlados revelam que ao estarem sujeitos a 400 milissegundos de espera realizaram menos 0.44% pesquisas nas primeiras três semanas e menos 0.77% nas segundas três semanas. Após os testes, os utilizadores levaram tempo até voltar a ter o mesmo número de pesquisas que tinham antes de os realizar. O tempo de espera para executar uma tarefa diminui a repetibilidade da sua execução. Os *websites* com tempos maiores de espera têm uma menor frequência de acesso por parte utilizadores depois da primeira visita.

A Google tem em conta a rapidez do *website*, no seu algoritmo de classificação dos resultados de pesquisa [3]. Um requisito do motor de busca é fornecer a melhor experiência de utilização, e a rapidez de acesso contribui para ela. A Microsoft conduziu um estudo [10] que consistia em analisar como os utilizadores se relembravam dos resultados depois de fazer uma pesquisa. Os utilizadores fizeram a pesquisa e meia hora mais tarde foi-lhes fornecido um inquérito. Nele era pedido para indicar os resultados da pesquisa que se lembravam. No final, comprovou-se que os resultados com melhores posições no ranking do motor de busca, eram os indicados com mais frequência no inquérito. Melhorar o tempo de acesso melhora a posição nos motores de busca e consequentemente aumenta o número de visitas.

Em 2012, a Walmart aprensentou um relatório acerca do impacto das vendas *online* em relação à performance do seu *website* [12]. Os resultados demonstraram o aumento de 2% da taxa de conversões, isto é, a taxa de aquisições online, por cada segundo a menos no tempo de acesso ao seu website. O aumento de 1% das receitas coincide com a diminuição de 100 milissegundos no tempo de espera de acesso. O desempenho das aplicações web contribui para o sucesso económico.

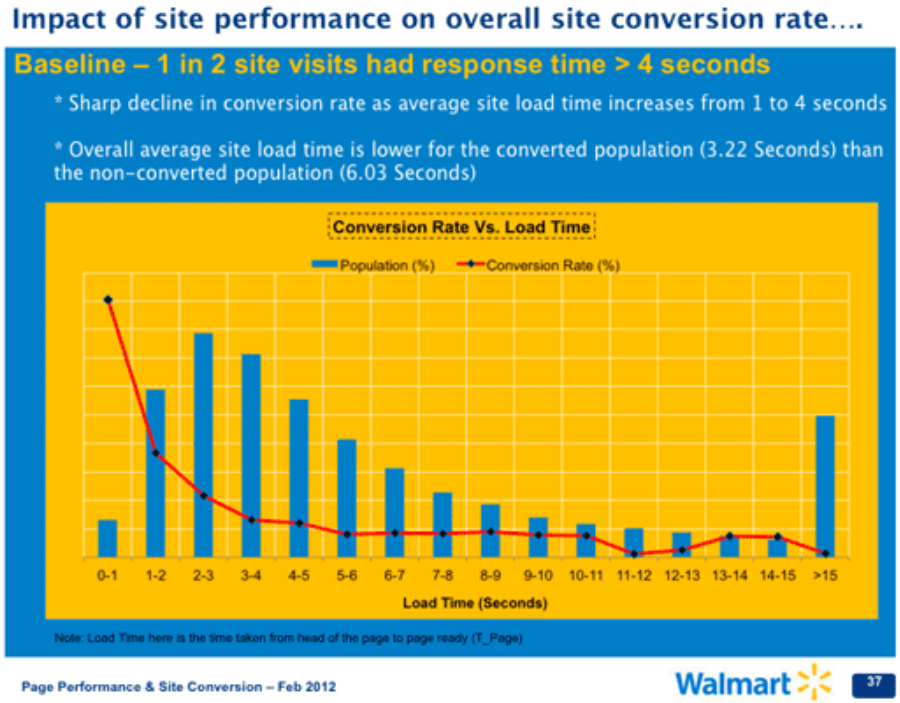


Figura 3: Taxa de conversões por tempo de carregamento [12]

Segundo a Figura 3 presente no relatório da Walmart, o acesso a páginas com tempo de espera entre 0 e 1 segundo resultaram numa taxa de aquisições *online* por acesso de 90%. A taxa decresce com o aumento do tempo de espera, e para tempos superiores a 15 segundos a taxa de aquisições é praticamente nula.

Os resultados dos estudos demonstram que a performance não é apenas um aspeto psicológico de necessidade de rapidez e demonstram empiricamente o seu impacto nos acesso aos *websites*. Face aos estudos analisados, os *websites* com melhores tempos de resposta têm mais sucesso porque:

- Têm uma taxa de rejeição menor. Há uma menor probabilidade dos utilizadores procurarem alternativas, sem sequer chegarem a ver o conteúdo do *website*;
- Têm mais confiança por parte da comunidade *online* por causa da boa experiência de utilização da sua plataforma Web;
- Têm um maior retorno de utilizadores, isto é, os utilizadores tornam a usar a plataforma após a primeira visita;
- Estão melhor classificados nas pesquisas dos motores de busca e por isso têm mais probabilidade de receber mais visitas;
- Convertem as suas visitas em lucros.

3 Medição

A realização de um inquérito a um conjunto de pessoas para classificar um *website* quanto à sua rapidez de carregamento não é suficiente. O tempo de espera é mensurável, e sem a sua medição é impossível tirar conclusões acerca do desempenho do *website* e dos resultados sobre o impacto de melhorias implementadas. A medição da performance é um aspeto importante a analisar nas organizações, pelas seguintes razões [13]:

- Estabelecer linhas base. Estas indicam os limites dos valores das métricas de performance que não devem ser ultrapassados;
- Detetar e resolver erros antes que estes tenham ocorrido num grupo maior de utilizadores;
- Medir a eficiência de alterações no código da aplicação. Um *website* está em constante modificação e a monitorização ajuda a detetar desde cedo problemas de performance;
- Medir impacto de alguma condição externa, como por exemplo, as falhas ou latência da rede;
- Resolver disputas com os utilizadores finais. Ao haver obrigações contratuais, os consumidores podem queixar-se da qualidade do serviço. A empresa pode provar que um *website* tem bom desempenho com os relatórios de performance e evitar reembolsos;
- Estimar a capacidade física (*hardware*) que é precisa no futuro. A performance está diretamente relacionada com o tráfego de rede. A análise dos dados recolhidos sobre a performance permite prever com mais rigor quando vai ser preciso melhorar as estruturas de rede.

Apresentam-se neste capítulo os processos associados à navegação Web que auxiliam a compreensão dos termos relacionados com a medição de desempenho. Identificam-se as métricas a analisar na avaliação da performance e os métodos existentes para recolher dados de desempenho.

3.1 Navegação na Web

O acesso rápido a uma página Web é condicionado por vários fatores. Conhecer os vários processos que ocorrem no pedido de uma página ajuda a compreender onde o tempo de espera de acesso é passado. Apresentam-se a seguir os vários processos que ocorrem na solicitação de uma página Web.

A navegação na Web é feita a partir do *browser*. Identifica-se a seguir cada uma dos seus módulos e descrevem-se as ações que ocorrem em cada um deles. Os mecanismos que estão por trás de cada módulo são descritos ordenadamente, desde a solicitação duma página Web até ela ser visualizada.

Segundo Tom Baker [14], a arquitetura do *browser* consiste em quatro camadas interligadas entre si que tornam possível a navegação na Web.

- A **camada de interface** fornece os controlos a partir dos quais o utilizador interage com o *browser*. Os controlos são a barra de localização, onde se escreve o URL que se pretende aceder e os botões de interação. Alguns destes botões servem para pedir novamente a página visualizada, navegar pelas páginas anteriormente acedidas e mudar configurações do *browser*.
- A **camada de rede** suporta as ligações aos servidores e a comunicação com eles. Ela realiza as procuras DNS, estabelece as conexões TCP, recebe, trata e envia pedidos HTTP.
- A **camada de motor de renderização** interpreta as páginas HTML, os ficheiros de estilos CSS e renderiza o conteúdo. Ela estrutura e aplica propriedades aos elementos da página através de um processo denominado caminho essencial de processamento.
- A **camada de motor de javascript** interpreta os ficheiros javascript e executa o código. Nesta camada adiciona-se interatividade aos elementos da página, associando-lhes eventos que alteram a sua estrutura e as suas propriedades através da API DOM. Esta API fornece instruções javascript utilizadas para alterar dinamicamente o conteúdo, a estrutura e as propriedades dos componentes da página.

A navegação inicia-se pela utilização da interface do *browser*. A escrita do URL da aplicação na barra de localização ou o clique em alguma hiperligação que está na barra de favoritos, no histórico ou numa página HTML tornam possível o acesso a uma página Web. O pedido da página Web a aceder desencadeia o início dos mecanismos realizados na camada de rede, esquematizados na Figura 4.

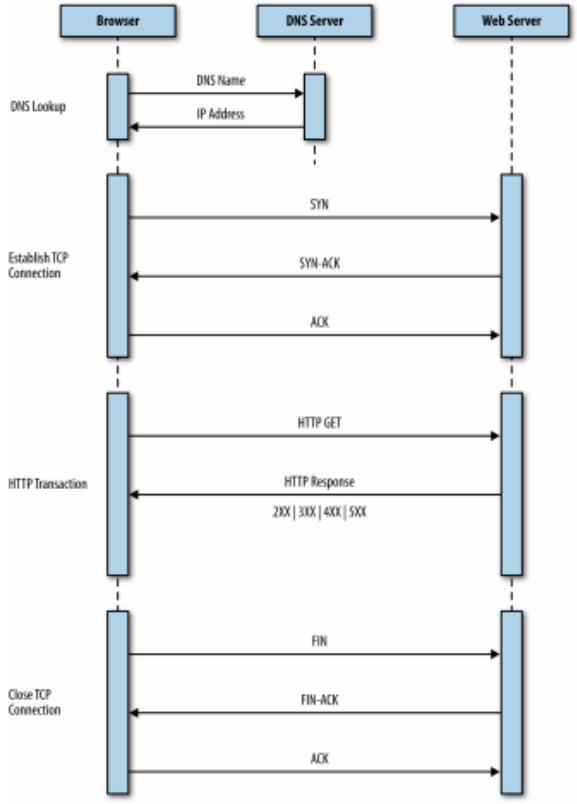


Figura 4: Processos de rede no browser [14]

O primeiro pedido de rede é feito a um servidor DNS. Nele é enviado o URL (por exemplo *www.google.com*) do recurso, com o qual o servidor identifica o domínio e responde ao cliente com o identificador de rede (IP) da máquina que tem o recurso pedido. Apenas com este identificador, os pedidos feitos pelos clientes seguem o caminho certo pela rede, até à máquina que fornece os recursos da aplicação. Os clientes após receberem o endereço IP, armazenam-no em *cache.* Uma vez armazenado em *cache,* nos pedidos de rede seguintes à mesma máquina, o servidor DNS não é consultado.

O conhecimento da localização da máquina através do IP desencadeia a tentativa de estabelecer a ligação do cliente com servidor, recorrendo ao protocolo TCP. O protocolo inicia a conexão através do método *three-way handshake*. No método, o cliente manifesta o interesse na conexão, o servidor indica ao cliente o conhecimento do seu interesse e a sua disponibilidade para prosseguir com a ligação e por fim o cliente liga-se ao servidor. No estabelecimento da conexão são definidas os parâmetros que permitem transmitir informação sem perdas. A conexão TCP persistente permite que sejam realizados vários pedidos de rede através da mesma conexão, evitando que ela tenha de ser estabelecida cada vez que o cliente pede um recurso ao servidor. O HTTP é o protocolo de comunicação utilizado na Web para fazer os pedidos de rede.

Após a negociação concluída é solicitado o recurso ao servidor através de um pedido de rede HTTP. O servidor nem sempre responde com o recurso pedido porque ele não existe ou não tem autorização para envia-lo, por exemplo. Os pedidos HTTP contêm um conjunto de informação sobre o *browser,* o recurso pedido e o servidor. Esta informação está dividida em cabeçalhos e um deles tem um código numérico de três dígitos que indica se a resposta enviada

ao cliente é o recurso pedido ou a razão de não ser enviado. Os códigos do estado dos pedidos estão numa das cinco gamas seguintes [15].

- 100-199 – indica uma resposta provisória;
- 200-299 – indica que o recurso pedido é enviado na resposta;
- 300-399 – indica que o cliente tem que fazer um pedido adicional para receber a resposta. Por exemplo, o código 301 avisa que o recurso pretendido está noutra máquina, realizando-se num novo pedido de rede;
- 400-499 – indica que há algo de errado nos parâmetros do pedido recebido pelo servidor. As razões podem ser, por exemplo, o recurso não existir ou não haver autorização para enviá-lo;
- 500-599 – avisa que ocorreu um erro no servidor e que não é possível de tratar o pedido.

Os recursos recebidos podem ser páginas HTML, ficheiros CSS, *scripts*, imagens, documentos JSON ou XML, PDFs, etc. No caso de ser uma página HTML, a camada de renderização inicia o caminho de processamento essencial. Este processo consiste nas próximas etapas [16]:

1. A página HTML é interpretada, extraindo as *tags* (por exemplo <html>) existentes e estabelecendo relações entre eles, formando uma estrutura em árvore. A estrutura é denominada DOM e representa o modelo de objetos do documento. A Figura 5 é um exemplo de uma árvore DOM. Numa página HTML podem ser encontradas *tags* que desencadeiam os pedidos de rede dos outros recursos, que podem ser por exemplo, ficheiros de estilos, javascript ou imagens. Nestes casos são desencadeados novos pedidos de rede.

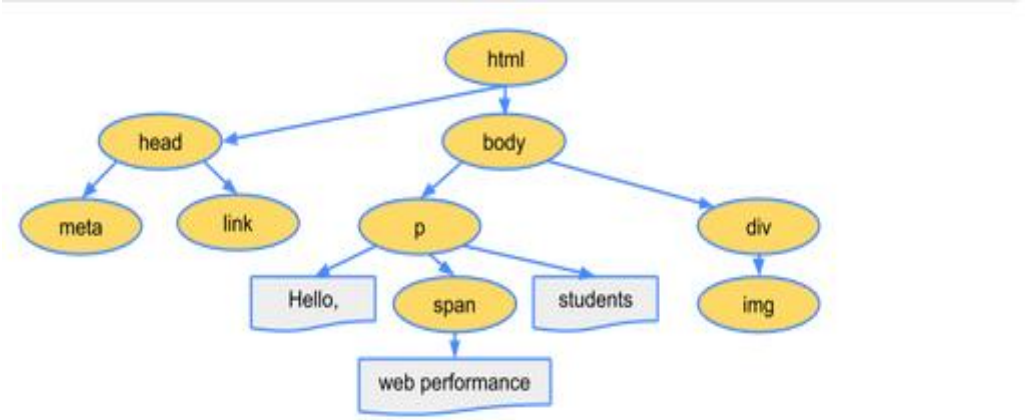


Figura 5: Árvore DOM [16]

2. Analogamente ao que acontece com os elementos HTML, as regras de CSS também são vinculadas a uma árvore chamada CSSOM, como a representada na Figura 6. A estrutura em árvore utilizada para mapear as propriedades tem uma razão de ser. As primeiras propriedades a serem associadas aos elementos são as dos estilos associados aos nós mais cimeiros que também são herdadas pelos nós filhos. As propriedades dos nós mais baixos da árvore prevalecem sobre as dos nós pai.

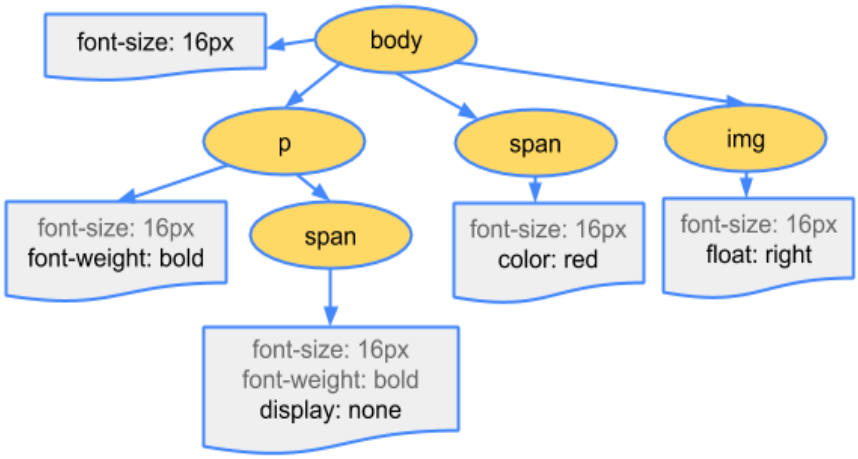


Figura 6: Árvore CSSOM [16]

3. As árvores DOM e CSSOM são combinadas em uma árvore de renderização, como a esquematizado na Figura 7. Nela apenas estão presentes os elementos que são desenhados. Os elementos invisíveis não estão presentes nela. Por exemplo, o nó "span", da árvore DOM da Figura 7, tem a propriedade "display" com o valor "none", que o torna invisível, fazendo com que não esteja presente na árvore de renderização (*Render Tree*).

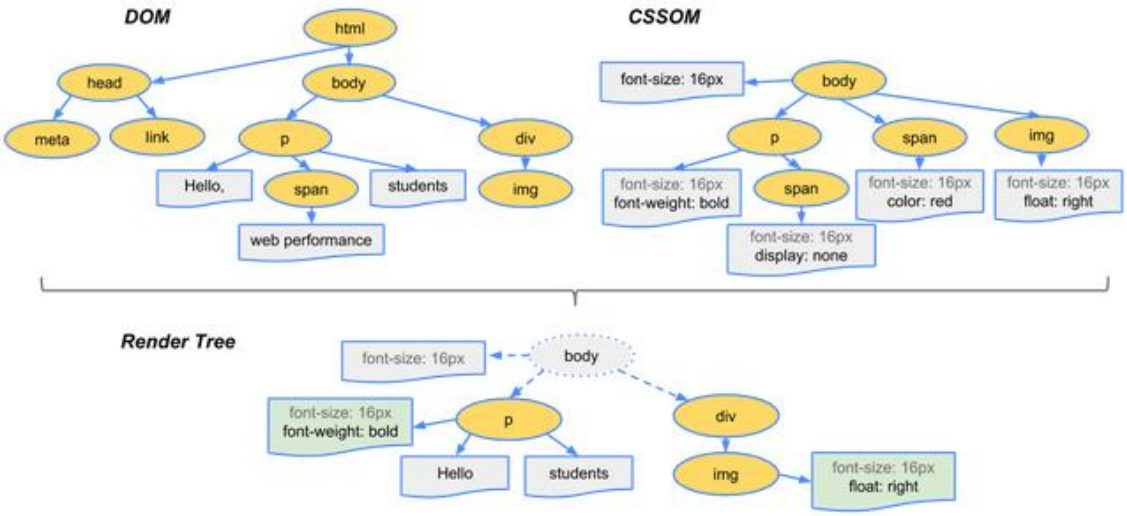


Figura 7: Árvore de renderização [16]

4. O processo seguinte tem a designação de refluxo. Nele o tamanho e a posição dos nós são calculados tendo em conta o tamanho da janela onde vai ser visualizada a página. A

Figura 8 ilustra a disposição de dois elementos na página HTML, tendo em conta o tamanho da janela de visualização.

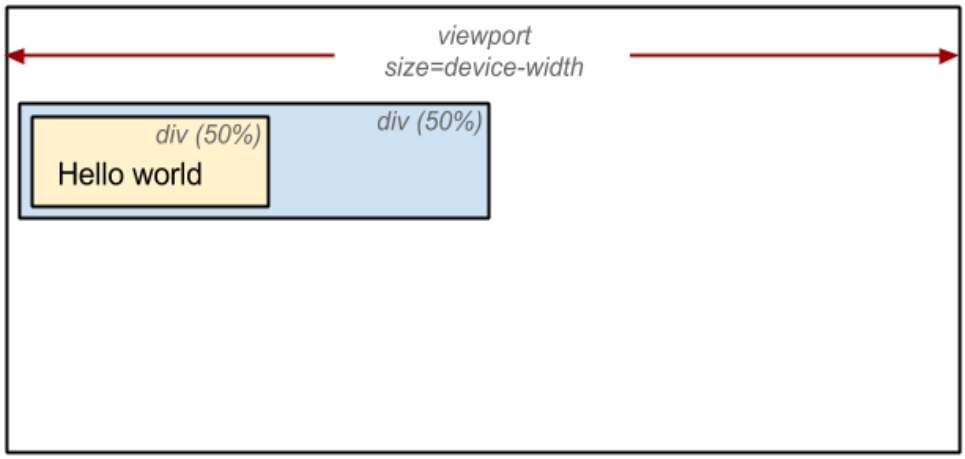


Figura 8: Refluxo [18]

5. Por fim, os nós passam a ser visíveis pelo utilizador, sendo convertidos em pixéis na janela de visualização. Esta etapa chama-se pintura.

Uma modificação nos elementos ou estilos a serem aplicados, faz com que estas cinco etapas tenham que ser novamente executadas. O *browser* tenta otimizar o seu processamento. Por exemplo, a mudança de cor de um elemento implica que seja executada apenas a tarefa 5 de pintura, enquanto que o redimensionamento de um elemento obriga a retomar o processo a partir da etapa 4.

Devido a mecanismos de otimização de processamento adotados pelo *browser*, existem elementos bloqueadores de renderização. A existência de ficheiros externos de estilos atrasa a construção da árvore CSSOM. O *browser* evita a sua construção múltiplas vezes. Ele espera que todos os ficheiros de estilos presentes na página sejam interpretados, por isso, é boa prática colocar a referência desses ficheiros no topo da página para que sejam interpretados o quanto antes. Os ficheiros javascript também são um elemento bloqueador porque conseguem alterar a árvore DOM e CSSOM. Este tipo de recurso deve ser colocado no fundo da página porque a renderização mantém-se bloqueada enquanto ele é transferido e executado.

3.2 Métricas

A performance na Web é avaliada pela medição do tempo desde que um utilizador pede algum recurso até ele estar disponível para ser utilizado. O utilizador ao aceder a uma página Web com bom desempenho, não tem a perceção do tempo esperado até ela estar disponível. A avaliação qualitativa da performance é medida através da perceção da velocidade que os utilizadores têm do tempo que a página Web levou a estar disponível. Quantitativamente, mede-se o tempo de carregamento de uma página, que corresponde ao tempo de *download* de todos os recursos e à visualização completa do seu conteúdo. Há outros indicadores que influenciam o tempo de carregamento da página, são eles:

- o **número de pedidos HTTP**. Por cada pedido de rede é aumentado o tempo de espera devido à latência da comunicação. A latência varia de acordo com o tipo de ligação, congestionamento da rede e outros fatores.
- o **tamanho do conteúdo transferido**. Quanto maior for, mais tempo demora a ser transferido pelo cliente. O total de bytes transferidos inclui o conjunto de ficheiros HTML, CSS, javascript, imagens, cookies e dados JSON.

Os peritos em performance Web Tammy Everts e Cliff Crocker apresentaram numa palestra [19] vinte métricas, representadas na Figura 9, a ter em conta na análise da performance.

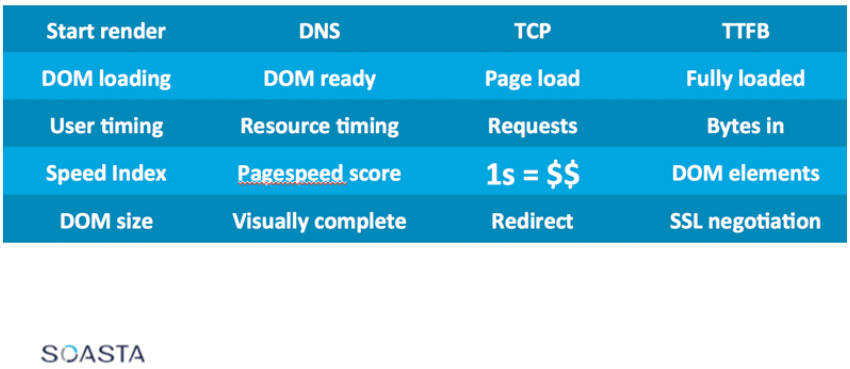


Figura 9: Métricas de performance [19]

Existem diferentes perfis de pessoas, dentro de uma equipa de desenvolvimento, com a responsabilidade de tornar um *website* o mais rápido possível. A diversidade de parâmetros a ter em conta na avaliação da performance torna difícil a sua escolha e a separação de responsabilidades. A seguir apresentam-se as métricas de performance, agrupadas por caso de uso, a serem avaliadas.

3.2.1 Medir a performance da rede

As operações de rede são as primeiras a serem efetuadas no acesso a uma página Web. As métricas a avaliar são as seguintes:

- Tempo de pedido do endereço IP de um domínio ao **DNS**, que pode ser zero se estiver armazenado em cache;
- Tempo de estabelecimento de conexão **TCP**, que será inexistente se houver a reutilização de uma conexão existente;
- Tempo de receber o primeiro byte de informação da resposta processada pelo servidor após o estabelecimento da conexão (**TTFB**).

Os dados disponibilizados permitem medir o impacto de ataques de negação de serviço na experiência dos utilizadores, analisar a eficiência do estabelecimento de conexão TCP e identificar problemas de DNS e atrasos no processamento das respostas no servidor. As métricas avaliam o desempenho do *backend*, ou seja, os tempos das operações que ocorrem na rede e no servidor.

3.2.2 Medir a performance dos eventos de *Frontend*

As operações que ocorrem no *frontend*, iniciam-se quando a página HTML é recebida no *browser*. A página é interpretada, sendo renderizada e transferidos recursos associados a ela. As métricas avaliadas são a complexidade da página HTML em termos do seu número de elementos e os tempos de eventos detetados no *browser*.

- **Dom loading**, evento onde o *browser* está prestes a analisar os primeiros bytes do documento HTML;
- **Dom ready,** quando o processamento da página foi concluído e todos os recursos presentes na pagina HTML foram transferidos;
- **Page load,** após o processamento completo da página é desencadeado o evento javascript **onLoad**, o momento que o *browser* determina que a página está pronta a ser utilizada;
- **Fully Loaded,** o momento em que os pedidos de rede cessam. A partir dos ficheiros javascript pode existir carregamento dinâmico de mais recursos;
- **Número de elementos do DOM**, quantos mais tiver, mais tempo demorará o processo de renderização.

3.2.3 Medir a performance por recurso

O conteúdo de terceiros revela-se muito importante para algumas aplicações web. Entre esse conteúdo podem estar presentes, por exemplo, anúncios, scripts de análise de tráfego, plugins de redes sociais e fontes externas de dados. Tudo o que é incluído numa página Web tem impacto no tempo de carregamento dela. A deteção de problemas de desempenho passa, por vezes, pela análise da duração de transferência de cada recurso incluído na página Web. Interessa detetar em todos os recursos presentes:

- A **duração da sua transferência**;
- O tempo da procura **DNS**;
- A conexão **TCP**;
- **O tempo de espera**, desde que é pedido o recurso até começar a ser obtido;
- **O tempo de transferência**.

Adicionar determinado conteúdo publicitário será benéfico para uma entidade, dependendo do aumento do tempo de carregamento do acesso à página Web. Como referido anteriormente, o elevado tempo de resposta pode diminuir as interações do utilizador na plataforma e reduzir as vendas. Interessa portanto avaliar o impacto de conteúdo externo na performance, os tempos de *download* por domínio e estudar a possibilidade de utilizar vários servidores que distribuam os recursos (CDN), para aumentar a proximidade geográfica aos utilizadores.

3.2.4 Medir a experiência de utilização

O mais importante das melhorias de desempenho é a melhoria da interação humana com as aplicações Web. Independentemente dos valores das outras métricas referidas, o utilizador não deve sentir-se frustrado na navegação Web.

Uma das métricas a medir é o início da renderização. Quanto mais cedo for iniciada, mais cedo o utilizador tem a perceção que a página existe. É fundamental que o utilizador veja progresso na renderização da página para não ter a sensação que ocorreu algum erro no sistema e desista de lhe aceder.

Algumas ferramentas importantes na análise da renderização das páginas Web, passam por captar várias fotografias no decorrer do carregamento da página. A Figura 10 retrata um vídeo gerado a partir da ferramenta *online* webpagetest.org. Num teste feito ao website viseuparticipa.pt, após 3.7 segundos, a página visualizada teve o aspeto representado na Figura 10.

Figura 10: Vídeo do carregamento de uma página Web

Outro aspeto a ter em conta é o momento em que a página se encontra visualmente completa. O utilizador tem a perceção que a página está pronta a ser utilizada quando são visualizados todos os elementos visíveis na sua janela de visão. A métrica que ilustra este momento é o *Speed Index* [20]. Este é expresso em milissegundos e depende do tamanho da janela de visualização (*viewport*). O acesso a uma página Web num telemóvel e num portátil são diferentes, pois o tamanho da janela de visualização no portátil é superior. A comparação de experiências de utilização de páginas de *websites* diferentes é um caso de uso desta métrica. A necessidade de utilização desta métrica surge da distorção da verdadeira experiência de utilização na análise do tempo de carregamento. A performance é avaliada em termos de tempos detetados pelo *browser* que não coincidem com o que o utilizador perceciona. A página HTML vai sendo interpretada e crescendo em altura, e enquanto vão sendo renderizados os elementos não visíveis, o tempo de carregamento continua a ser contabilizado. Por exemplo, o evento "onLoad" do *browser* pode indicar que a página demorou 7 segundos a carregar, mas se o conteúdo presente na janela de visualização ficar preenchido totalmente em 2 segundos, a impressão que o utilizador vai ter, é que a página demorou apenas 2 segundos a ficar disponível.

Existem elementos prioritários que requerem um carregamento mais rápido que outros. Por exemplo, num *website* de notícias será interessante surgir a notícia com mais destaque o mais rápido possível para que capte a atenção do utilizador, sem que ele dê pela falta dos outros

elementos a serem renderizados. O que se pretende com esta situação é manter o utilizador ocupado para reduzir a sua perceção do tempo de espera.

3.2.5 Medir impacto da performance no negócio

O gestor de negócio procura significado dos tempos de carregamentos em termos monetários, ou seja, os benefícios da performance para os indicadores principais do negócio, que podem ser, por exemplo, as vendas *online*. A comparação da performance com a sua concorrência mais direta é uma mais-valia para definir prioridades no desenvolvimento da aplicação. A Figura 11 representa a taxa de conversão de uma transação de um negócio em comparação com o tempo de total de espera. Conclui-se que para tempos menores de espera há maiores taxas de conversão.

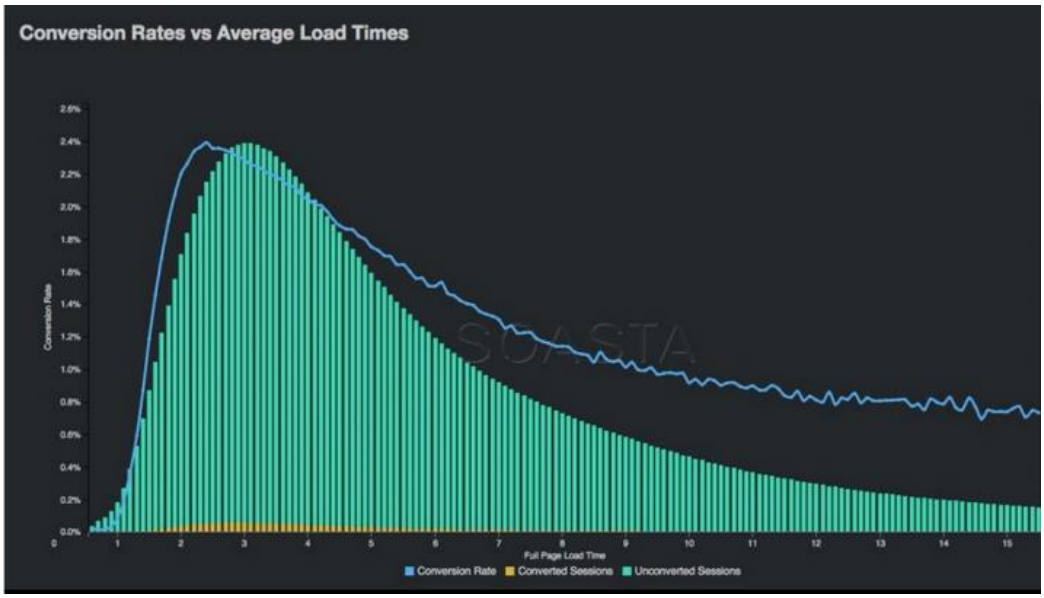


Figura 11: Taxa de conversões Vs Tempo médio de carregamento [19]

A avaliação da performance é um trabalho de equipa que requer as aptidões de vários profissionais e a colaboração entre eles.

3.3 Técnicas de medição

Duas formas de medir a performance das aplicações Web passam pela monitorização de utilizadores reais e a realização de testes de performance efetuados por máquinas, denominados testes sintéticos.

3.3.1 Monitorização de utilizadores reais

A monitorização de utilizadores reais (RUM) é uma técnica passiva utilizada para recolher dados dos utilizadores sobre a sua experiência de utilização em páginas Web. O relacionamento de dados de performance com os dados de utilização do *website*, como por exemplo, o

relacionamento do número de visitantes com o tempo médio por página, permitem avaliar o impacto da performance no comportamento dos utilizadores.

Na monitorização recolhem-se dados de performance que são utilizados para verificar se as páginas Web são entregues rapidamente e sem erros. Existe um conjunto diverso de variáveis que influenciam a rapidez com que os utilizadores navegam na Web. Uma plataforma Web é acedida em diferentes condições de rede, *browsers*, sítios e dispositivos. A variedade de ambientes permite identificar problemas de performance e com base na análise das suas características podem encontram-se soluções.

Tratando-se de uma técnica passiva, a recolha de dados é induzida pelos utilizadores quando acedem a uma página Web monitorizada. As medições de desempenho no servidor não têm em consideração a latência da rede do cliente, portanto os tempos aí recolhidos não avaliam com precisão a experiência de utilização. O servidor contabiliza o tempo que demora a tratar os pedidos de rede dos clientes, mas não regista os tempos passados na procura DNS, no estabelecimento de conexão TCP e noutros eventos do *frontend*. A medição deve ser feita do lado do cliente. Uma solução de recolha de dados passa pela injeção de código javascript que recolhe os tempos de duração dos eventos que ocorrem no *browser* e depois reporta-os ao servidor.

3.3.1.1 Navigation Timing API

O "W3C Web Performance Working Group" padronizou a *Navigation Timing API* [21], capaz de medir eventos ligados ao carregamento da página Web. A API está presente nos *browsers* com maior quota de utilizadores. Segundo a Figura 12, não é suportada por versões anteriores à versão 9 do Internet Explorer e versão 9 do Safari. Nos dispositivos móveis não é suportada pelo iOS Safari e Opera Mini. A recolha de dados de utilizadores tem limitações devido ao suporte dos *browsers* utilizados e nos casos em que o javascript não está ativo.

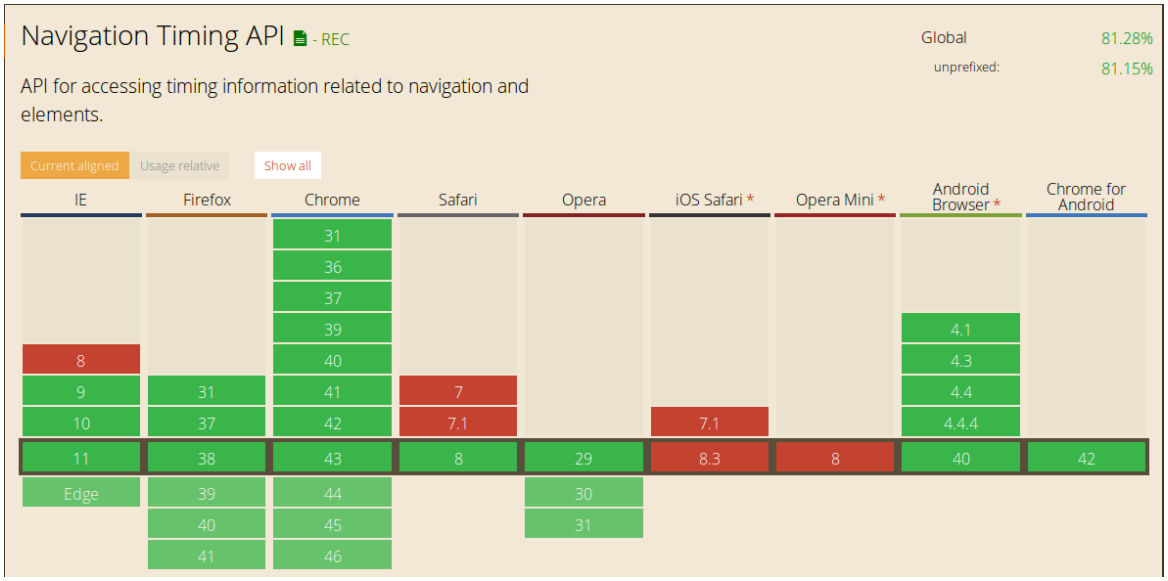


Figura 12: Suporte à Navigation Timing API [22]

A seguir documenta-se como podem ser acedidos os dados de performance recolhidos pela API.

No acesso à consola de um *browser* podem ser visualizados dados de performance da página acedida, escrevendo o comando “performance.timing”, como se apresenta na Figura 13.

.

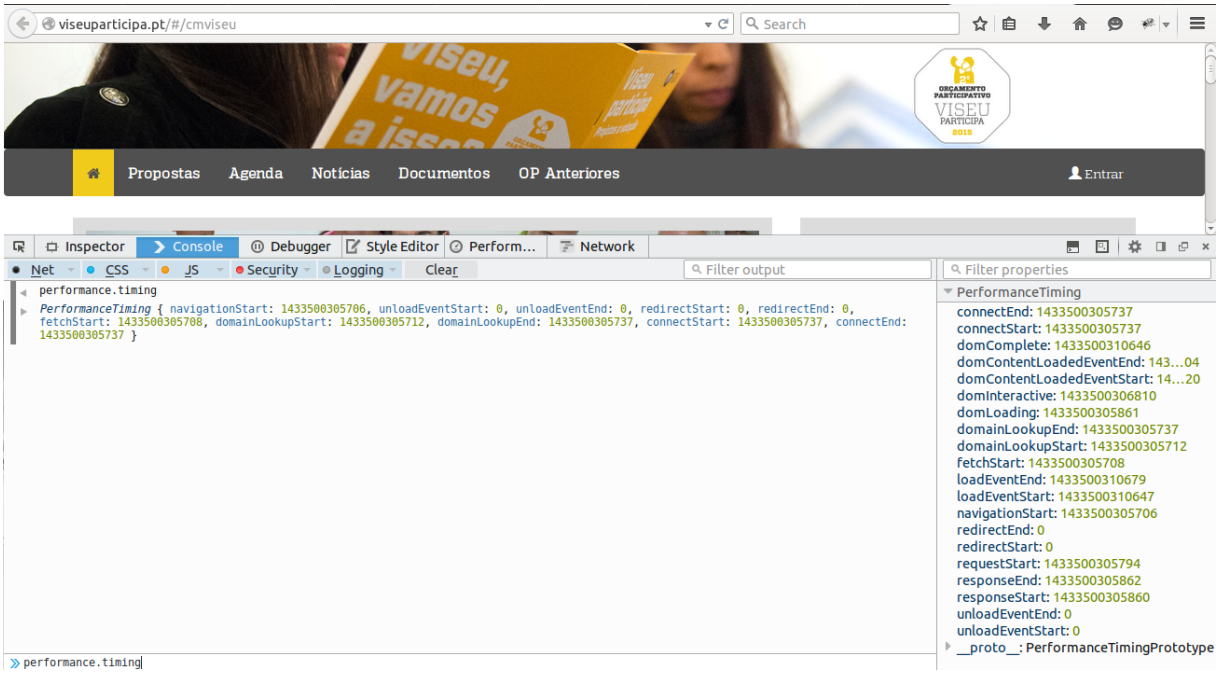


Figura 13: Momentos do carregamento de uma página Web

Na Figura 13, do lado direito, por baixo do título “PerformanceTiming” apresentam-se os momentos em que ocorrem vários eventos no carregamento da página. Por exemplo, os eventos “connectEnd” e “connectStart”, dizem respeito ao fim e início do estabelecimento da conexão TCP. Os seus valores são dados pelo instante em que ocorrem, em número de milissegundos, desde 1 de Janeiro de 1970 UTC [23]. Os eventos monitorizados estão representados na Figura 14.

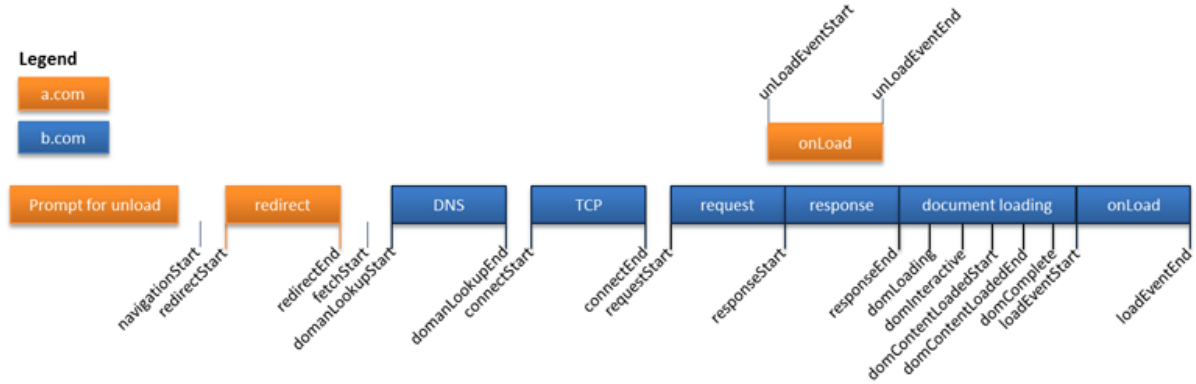


Figura 14: Processos de navegação na Web [24]

Na Figura 14 são apresentados sequencialmente os eventos que ocorrem na transição entre duas páginas. A cor-de-laranja estão os eventos que ocorrem na página inicial e a azul os eventos da página à qual se acede. Inicialmente ocorre o *unload* do DOM da página inicial, ou

seja, os elementos da página atual desaparecem. A seguir inicia-se a navegação. Podem existir redireccionamentos dependendo da configuração do acesso à página no servidor. Após isto, podem ocorrer uma de duas coisas, no evento *fetchStart* é verificada na *cache* se o recurso pedido já existe e é iniciado o carregamento da página (*document loading*) ou é preciso obter o endereço IP a partir do DNS, estabelecer a ligação TCP e fazer o pedido HTTP para receber a página. No carregamento e interpretação do recurso ocorrem as etapas do caminho essencial de processamento, medidas a partir dos eventos seguintes [24]:

- **domLoading –** quando o *browser* inicia a interpretação dos primeiros bytes do documento HTML;
- **domInteractive -** marca o ponto em que o *browser* terminou de interpretar o documento HTML e a construção do DOM está completa;
- **domContentLoaded -** momento em que o DOM está completo e não há ficheiros de estilos a bloquear a execução do javascript, quer isto dizer que é possível iniciar a construção da árvore de renderização;
- **domComplete –** quando o processo de interpretação do documento HTML está completo e todos os recursos da página foram recebidos;
- **loadEvent –** o evento disparado quando termina o carregamento da página.

Tendo em conta os momentos em que ocorrem os eventos retiram-se as seguintes informações:

- **Tempo de procura DNS,** é dado pela diferença entre os momentos dos eventos *domainLookupEnd* e *domainLookupStart*;
- **Tempo de estabelecimento de conexão TCP,** é dado pela diferença entre os momentos dos eventos *connectEnd* e *connectStart*;
- **Tempo de espera para receber o primeiro byte após ter sido feito o pedido http (TTFB),** é dado pela diferença entre os momentos dos eventos *responseStart* e *connectEnd*;
- **Tempo de transferência do recurso,** é dado pela diferença entre os momentos dos eventos *responseEnd* e *responseStart*;
- **Tempo passado no Backend (latência),** é dado pela diferença entre os momentos dos eventos *responseEnd* e *fetchStart*;
- **Tempo passado no Frontend,** é dado pela diferença entre os momentos dos eventos *loadEventStart* e *responseEnd*;
- **Tempo de carregamento de página,** é dado pela diferença entre os momentos dos eventos *loadEventEnd* e *navigationStart*.

Os tempos de carregamento analisados dizem respeito ao ficheiro HTML, o primeiro recurso recebido quando se acede uma página web. A análise do desempenho de cada recurso transferido, através da utilização da API, pode ser efetuada na execução do comando "performance.getEntries()" na consola do *browser*.

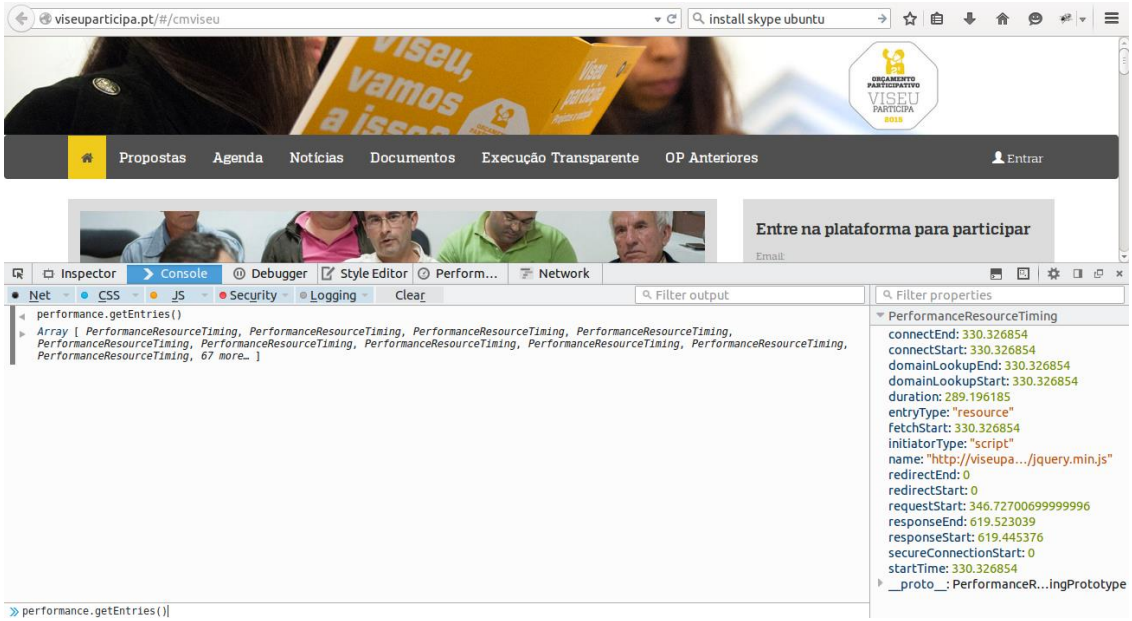


Figura 15: Momentos da transferência dum recurso

A Figura 15 apresenta o resultado da execução do comando "performance.getEntries()". O resultado é um vetor com os dados de desempenho dos recursos transferidos. A partir da informação devolvida identifica-se a duração dos vários eventos decorridos na transferência de cada recurso e a forma como é transferência é feita. Por exemplo, quando o valor de *domainLookupStart* e *domainLookupEnd* é o mesmo, como representado na Figura 15, do lado direito, conclui-se que não ocorreu nenhum pedido ao DNS e possivelmente o IP do domínio está guardado em cache. Nos valores de *connectStart* e *connectEnd* também ocorre o mesmo, não há estabelecimento de uma conexão porque o ficheiro está armazenado em cache ou ocorreu uma reutilização da ligação TCP. Na informação analisada identifica-se ainda o tipo de recurso e a partir de que domínio é transferido. A utilização destes dados permite medir o impacto dos recursos no tempo de carregamento da página, relacionando com o domínio e o tipo de recurso.

Existem algumas desvantagens na utilização de código javascript para realizar a monitorização.

- A recolha de dados obriga a que exista um envio dessa informação para algum servidor, o que afeta o desempenho da aplicação;
- Se o código de monitorização não for executado ou o utilizador desativar o javascript no *browser* não são recolhidos dados.

3.3.1.2 Single Page Application

A medição do desempenho do lado do cliente apresenta alguns desafios, devido à abordagem seguida na construção das aplicações Web. Associado à evolução da web de 1.0 para 2.0 está o aparecimento de novas tecnologias. O termo de evolução para Web 2.0, não foi bem aceite por Tim-Berners Lee [25], inventor da world wide web, porque segundo ele as várias tecnologias ditas novas, já teriam aparecido antes da própria Web. A necessidade de melhorar a experiência de utilização das aplicações, com interfaces mais rápidas e fáceis de usar, torna-se possível com a utilização de Ajax, padronizada em 2006 pela World Wide Web Consortium [26]. O Ajax permite fazer pedidos assíncronos de dados sem interferir com o comportamento

da página a partir da qual são feitos. A utilização destas novas tecnologias permitiu a utilização de uma nova arquitetura de construção de *websites*, que passaram a ser conhecidos por *single page applications* (SPA).

As aplicações Web surgem numa primeira fase como um conjunto de páginas estáticas ligadas entre si através de hiperligações. Mais tarde aparecem as linguagens de servidor que as tornam mais dinâmicas. As páginas passam a ser geradas dinamicamente no servidor e só depois enviadas para o cliente. As interações neste tipo de aplicações resultam no pedido de uma nova página ao servidor, que substitui aquela que está a ser visualizada.

O aparecimento das tecnologias Ajax e DHTML permitem que a lógica das aplicações passe a estar do lado do cliente, aliviando o processamento do servidor e diminuindo a quantidade de dados trocados.

O Ajax, o HTML5 e as aplicações Web *mobile* levaram ao crescimento da linguagem javascript. As aplicações centradas no cliente, são comummente designadas de aplicações de página única, mais conhecidas por SPA, pois o utilizador encontra-se sempre na mesma página, à medida que vai recebendo os componentes necessários. Por outras palavras, o conteúdo vai sendo atualizado de acordo com as ações do utilizador. São descarregados todos os componentes necessários da aplicação Web, num primeiro acesso. A seguir, a comunicação com o servidor passa pela utilização de pedidos Ajax, a uma API de dados que depois são carregados dinamicamente nas vistas HTML.

Num artigo de Nic Jansma [27], identificam-se três problemas na monitorização de utilizadores reais das SPA.

- **Deixa de haver interesse em medir a performance utilizando o evento javascript *onload***. Este evento ocorre quando todos os recursos da página Web são descarregados e supostamente os elementos HTML já estão desenhados. Uma vez que é possível fazer pedidos assíncronos nos ficheiros javascript descarregados, pode haver neles código que desencadeiam pedidos Ajax. Assim considera-se que o tempo de carregamento dado pelo evento *onload* não é o mais correto para medir a performance neste tipo de aplicações.

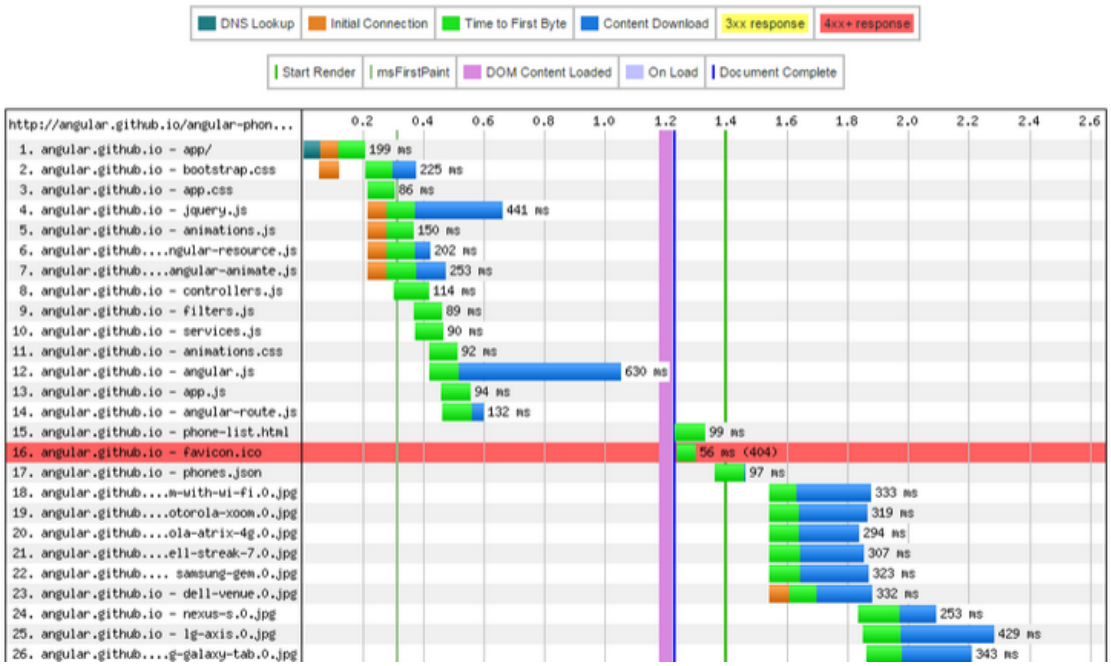


Figura 16: Carregamento de uma SPA [27]

A Figura 16 apresenta o gráfico em cascata de uma SPA. No carregamento da página, o evento "onLoad" ocorre aos 1,2 segundos, mas continuam a ser descarregados outros recursos.

- Como o próprio nome indica, *single page application*, **a navegação real só é feita a primeira vez que acedemos à aplicação**. Uma vez que é carregado e interpretado o conteúdo, a "navegação" para outra página implica mudar elementos HTML e realizar pedidos assíncronos para ir buscar dados para a preencher. Isto é, a medição da performance apenas é feita no primeiro acesso à aplicação.
- O terceiro problema é consequência dos dois primeiros, **o *browser* não informa de todos os recursos que foram descarregados**. Ao extrair dados sobre a performance ao ser executado o evento javascript *onload*, não obtém informação sobre quaisquer pedidos assíncronos feitos após ter sido executado. A navegação para outra página não obriga a executar esse mesmo evento, uma vez que não há um recarregamento total da página.

Os problemas mencionados obrigam a um esforço extra na programação do script que faz a monitorização dos dados. O tipo de aplicação é construído recorrendo normalmente às *frameworks javascript*, como por exemplo Angularjs, Emberjs e React. A recolha de dados de desempenho na mudança de páginas destas aplicações consiste em conhecer a *framework* e programar o evento da mudança de página, não detetada pelo *browser*, para enviar essa informação.

3.3.2 Testes sintéticos

Os testes sintéticos envolvem *bots* automáticos que executam instruções que avaliam a performance de uma página Web. Ao contrário do RUM, trata-se de uma recolha de dados induzidos que não segue ações de utilizadores reais.

Os testes podem ser simulados em distintos locais físicos, condições de redes, dispositivos e intervalos de tempo. Ao realizar testes sempre nas mesmas condições, existem grandes hipóteses dos resultados de performance variarem pouco entre si. A estabilidade fornecida por este tipo de testes, torna-os úteis na definição de valores limites de métricas de performance, permitindo comparar valores das métricas em diferentes versões da mesma aplicação. A otimização de algum aspeto da aplicação será melhor avaliado recorrendo aos resultados destes testes.

A execução periódica destes testes verifica a disponibilidade da aplicação, um requisito essencial para uma plataforma Web. Segundo a loja *online* Etsy [28], este tipo de testes é uma parte essencial do seu desenvolvimento, pois permite-lhes avaliar as tendências da performance do *frontend* na aplicação, antes de ser colocada em produção, eliminando dúvidas quanto à variabilidade da latência da rede sofrida pelos utilizadores reais.

A recolha de dados de performance seguindo esta técnica não envolve modificações na aplicação, ao contrário do que acontece em RUM. Esta característica permite que os mesmos testes sejam feitos a aplicações concorrentes para comparar os dados de performance.

O inconveniente desta técnica é o reduzido leque de ambientes simulados, restringidos a uma área geográfica e às condições de rede semelhantes. É possível simular os mais variados ambientes mas com um custo, que a certo ponto não será sustentável para a entidade interessada neles.

Os testes de monitorização de utilizadores reais têm os seguintes benefícios:

- Entender como a aplicação está a ser utilizada;
- Perceber a verdadeira distribuição geográfica dos utilizadores e avaliar o desempenho nas várias zonas geográficas;
- Compreender a distribuição da rede;
- Garantir a visualização completa da utilização da aplicação e a performance;

Os testes sintéticos têm os seguintes benefícios:

- Monitorização em ambiente controlado;
- Diagnosticar erros de acesso à aplicação mediante determinadas condições;
- Comparação de performance entre aplicações,
- Não é necessário modificar a aplicação a alterar, para recolher dados;
- Verificar disponibilidade da aplicação;
- Estabelecer limites para valores de métricas.

Nenhuma técnica é melhor que outra, cada abordagem se adequa mais a determinadas situações do que a outra. O RUM transmite a verdadeira experiência de utilização dos utilizadores reais enquanto os testes sintéticos dão maior detalhe acerca de problemas de performance da aplicação e informam sobre a sua disponibilidade.

4 Ferramentas de medição

A utilização de ferramentas de medição de performance permite analisar o impacto do tempo de espera na experiência de utilização de um *website.* Elas recolhem dados relacionados com as métricas apresentadas no capítulo anterior e representam a informação em elementos gráficos que facilitam a sua análise. A medição é um processo contínuo e Lara Hogan [10] sugere utilizar ferramentas, representadas na Tabela 1, com periodicidades diferentes.

Ferramenta	Tipo	Funcionalidades	Periodicidade
YSlow	Ferramenta de *browser*	Classificação geral e recomendações	Enquanto é feito o desenvolvimento e depois em cada trimestre.
Chrome DevTools	Ferramenta de *browser*	Recomendações, gráfico em cascata e *frames* por segundo	Enquanto é feito o desenvolvimento e depois em cada trimestre.
WebPagetest	Teste sintético	Classificação geral, recomendações, gráfico em cascata e *Speed Index*	Cada vez que é feita uma grande mudança.

Catchpoint, Gomez, wpt-script, etc.	Teste sintético	Tendências de performance no *website*	Mensalmente
Google Analytics, mPulse, Glimpse	Monitorização de utilizadores reais	Média de tempo de carregamento por área geográfica	Semanalmente

Tabela 1: Ferramentas de medição de performance e periodicidade de utilização

A seguir são apresentadas algumas das ferramentas presentes na Tabela 1. Na categoria de ferramentas de *browser* são apresentadas o YSlow e o Chrome DevTools. Demonstram-se as funcionalidades das ferramentas WebPagetest e SpeedCurve que executam testes sintéticos de performance e alguns projetos que utilizam Phantomjs para recolher sinteticamente os dados de performance. Na categoria de monitorização de utilizadores reais estão as plataformas Web Soasta mPulse e Google Analytics e outros projetos *open source*.

4.1 Ferramentas de *browser*

A análise de desempenho deve iniciar-se com a utilização de ferramentas de *browser* no decorrer do seu desenvolvimento. Estas ferramentas são muito úteis para fazer *debug* de código javascript, alterar propriedades CSS dos elementos HTML e analisar a performance. A seguir apresentam-se algumas ferramentas integradas e integráveis num *browser*.

4.1.1 YSlow

O YSlow é um *plugin* de *browser*, criado por Steve Souders, que pode ser instalado nos *browsers* Firefox, Opera, Chrome e Safari. Este *plugin* faz uma avaliação classificativa e dá recomendações de aspetos a serem melhorados nos *websites* que testa. Os critérios de avaliação utilizados focam-se na análise das medidas de otimização implementadas na aplicação, sugeridas no livro *High Performance Web Sites* [29], escrito pelo criador do plugin.

Os resultados de um teste estão divididos nas secções classificação, componentes e estatísticas.

Na Figura 17 está representada a secção de classificação, onde cada aspeto é classificado de “A” a “F”, sendo “A” o indicador da melhor classificação. São avaliados 23 aspetos que podem ser divididos nas categorias conteúdo, *cookie*, CSS, imagens, javascript e servidor.

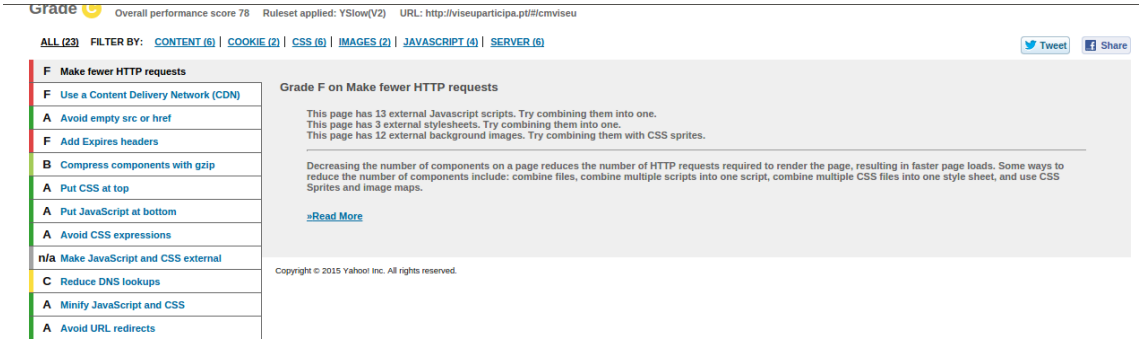


Figura 17: Secção de classificação do Yslow

A cada um dos aspetos está associada uma descrição do problema e uma solução de melhoria no caso de não ser obtida a melhor classificação. O programador conhece melhor que ninguém a aplicação que desenvolve e nem sempre é possível aplicar as recomendações sugeridas. Por exemplo, como se vê na Figura 18, é sugerido que para diminuir o número de pedidos de rede, deve-se tentar combinar os treze ficheiros javascript num só. Alguns destes scripts são provenientes de domínios diferentes, como por exemplo, o script da Google que monitoriza o número de visitas e os restantes scripts, o que torna impossível a combinação.

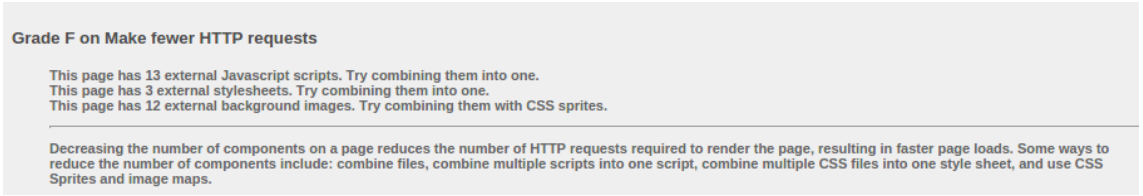
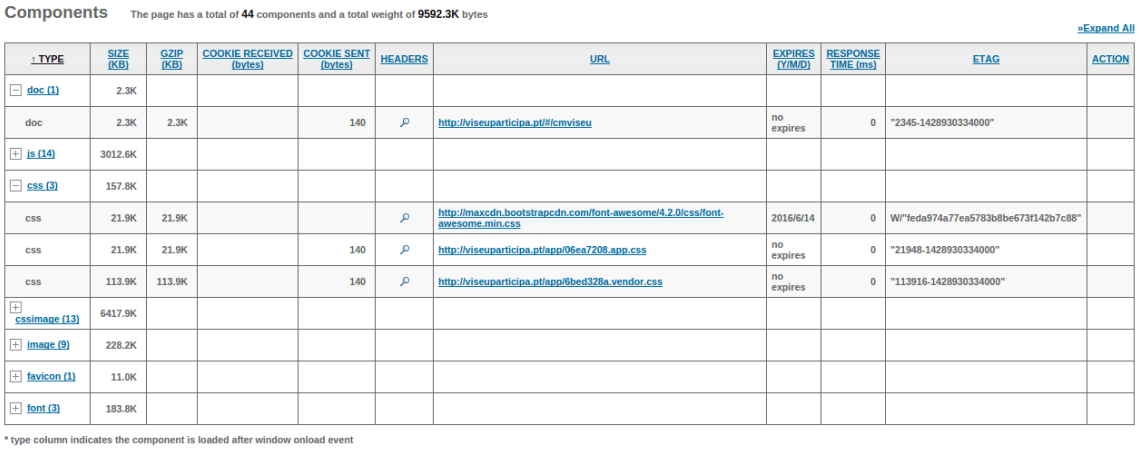


Figura 18: Recomendação do Yslow

Deve-se analisar o impacto e limitações para avaliar a possibilidade de implementar as recomendações.

Na secção dos componentes, são identificados o número de recursos e o seu tamanho em bytes. São também discriminadas várias informações sobre cada um dos componentes da página, como por exemplo, o tamanho, o URL, o tempo de expiração e o tempo de carregamento, como representa a Figura 19.



↑ TYPE	SIZE (KB)	GZIP (KB)	COOKIE RECEIVED (bytes)	COOKIE SENT (bytes)	HEADERS	URL	EXPIRES (Y/M/D)	RESPONSE TIME (ms)	ETAG	ACTION
⊟ doc (1)	2.3K									
doc	2.3K	2.3K		140		http://viseuparticipa.pt/#/cmviseu	no expires	0	"2345-1428930334000"	
⊞ js (14)	3012.6K									
⊟ css (3)	157.8K									
css	21.9K	21.9K				http://maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css	2016/6/14	0	W/"feda974a77ea5783b8be673f142b7c88"	
css	21.9K	21.9K		140		http://viseuparticipa.pt/app/06ea7208.app.css	no expires	0	"21948-1428930334000"	
css	113.9K	113.9K		140		http://viseuparticipa.pt/app/6bed328a.vendor.css	no expires	0	"113916-1428930334000"	
⊞ cssimage (13)	6417.9K									
⊞ image (9)	228.2K									
⊞ favicon (1)	11.0K									
⊞ font (3)	183.8K									

Figura 19: Tabela de recursos do Yslow

A Figura 20 ilustra a secção de estatísticas. O gráfico descreve o contributo de cada tipo de componente em termos do número de recursos e bytes descarregados, nas situações em que a *cache* está vazia e quando os componentes estão armazenados nela.

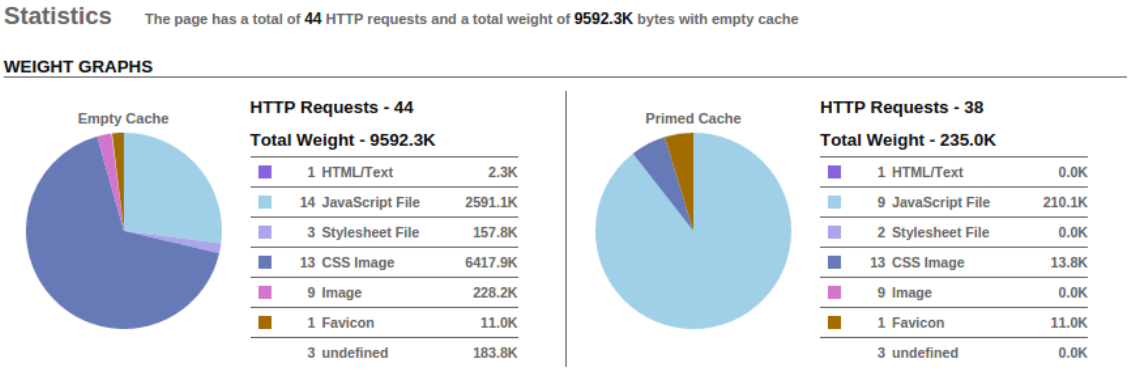


Figura 20: Análise de número e tamanho de recursos no Yslow

4.1.2 Chrome Devtools

O Chrome Devtools é um conjunto de ferramentas construídas no *browser* Chrome, com o objetivo de facilitar a análise da informação das aplicações Web. Além de ajudar no desenvolvimento, a ferramenta pode ser utilizada como auxílio na melhoria do desempenho.

À semelhança do YSlow, o Chrome Devtools também analisa a página e fornece alguns conselhos de melhoria de performance. A secção dedicada a esta funcionalidade tem o nome de "Audits" (auditoria).

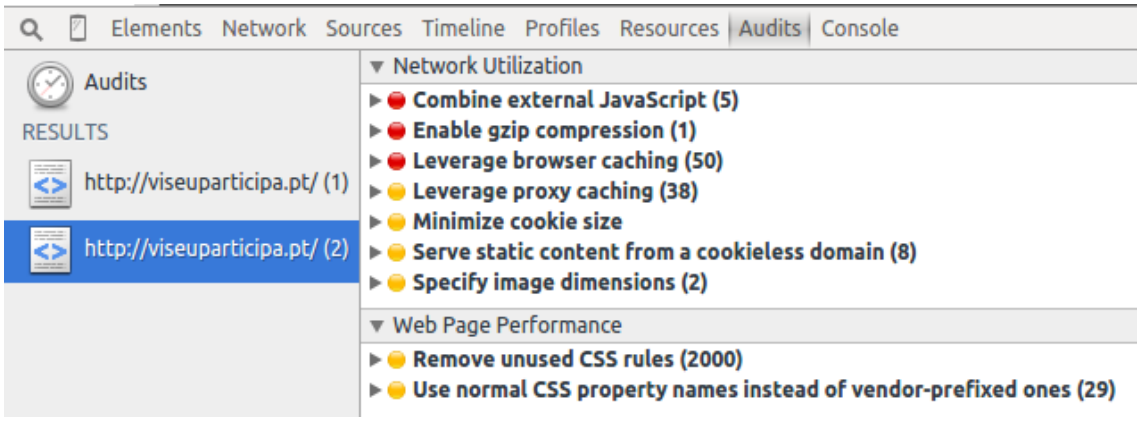


Figura 21: Secção de auditoria do Chrome DevTools

O teste desta funcionalidade foi feito à mesma página que em YSlow e, no relatório apresentado na Figura 21, recomenda-se uma vez mais combinar os vários recursos javascript.

Uma das formas de visualização mais importante na análise detalhada da performance é o gráfico em cascata. Este tipo de gráfico permite visualizar numa linha temporal todos os recursos carregados no acesso a uma página Web e disponibiliza os tempos demorados nos vários processos de transferência. No Chrome Devtools, o gráfico da Figura 22 pode ser visualizado na secção "Network" (Rede).

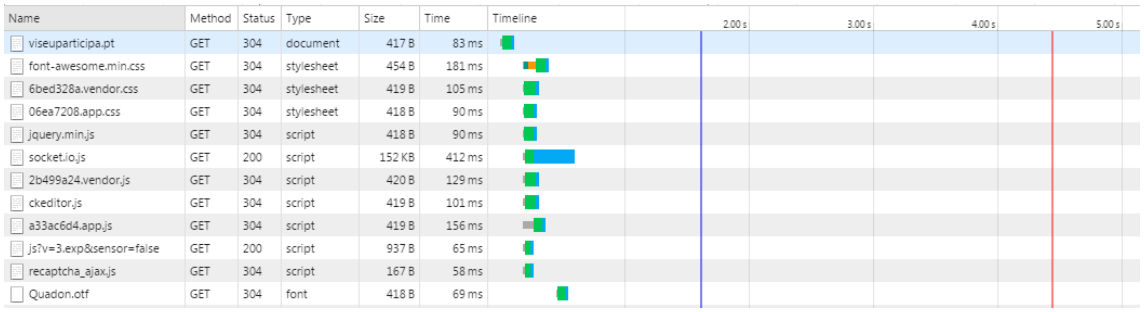


Figura 22: Secção de rede do Chrome Devtools

No gráfico em cascata visualizam-se os recursos que afetam mais o tempo de carregamento da página e identificam-se os recursos bloqueadores da transferência. No eixo vertical estão representados os vários recursos por ordem de início de transferência. No eixo horizontal visualiza-se o tempo de carregamento de cada recurso. Neste tempo é discriminado o contributo temporal da procura DNS, do estabelecimento da conexão TCP e de *download*. Aqui é possível verificar se existem recursos que bloqueiam o início da transferência de outros recursos.

A secção de análise de pedidos de rede também está presente nas ferramentas dos *browsers* Firefox, Internet Explorer, Opera e Safari.

4.2 Ferramentas que executam testes sintéticos

As ferramentas dos *browsers* revelam-se muito úteis no processo de desenvolvimento de uma aplicação. Elas permitem acompanhar minuciosamente o impacto dos vários recursos na página Web e aconselha soluções para melhorar a sua eficiência.

No desenvolvimento de uma aplicação Web, o programador utiliza a sua máquina como servidor por questões de flexibilidade. Na análise do gráfico em cascata com as ferramentas do *browser* não é detetado tempo passado na rede porque o servidor e cliente estão na mesma máquina. Portanto não é tida em conta a latência da rede no tempo de carregamento, o que não espelha a realidade dos utilizadores.

Os testes sintéticos são executados em ambientes controlados, nas mesmas condições, realizando uma avaliação mais rigorosa do impacto das alterações nas aplicações Web. A seguir apresentam-se ferramentas *online* e meios de executar testes sintéticos.

4.2.1 Webpagetest

O WebPageTest (http://www.webpagetest.org) é uma plataforma *online* que executa gratuitamente testes sintético a qualquer *website*, disponibilizando várias opções de configuração.

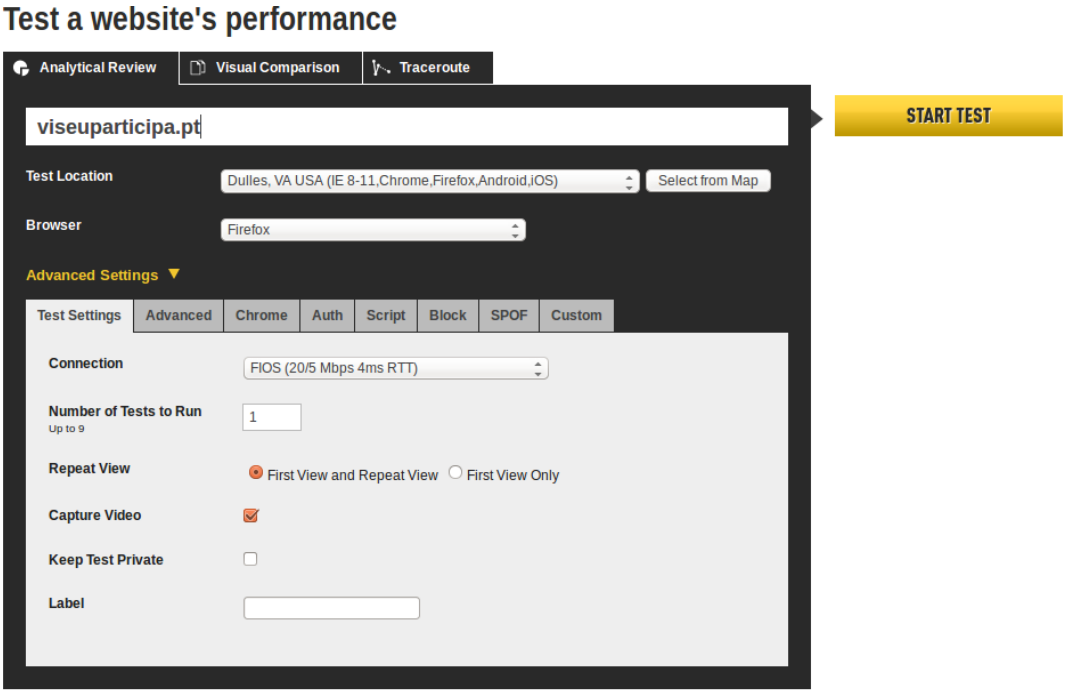
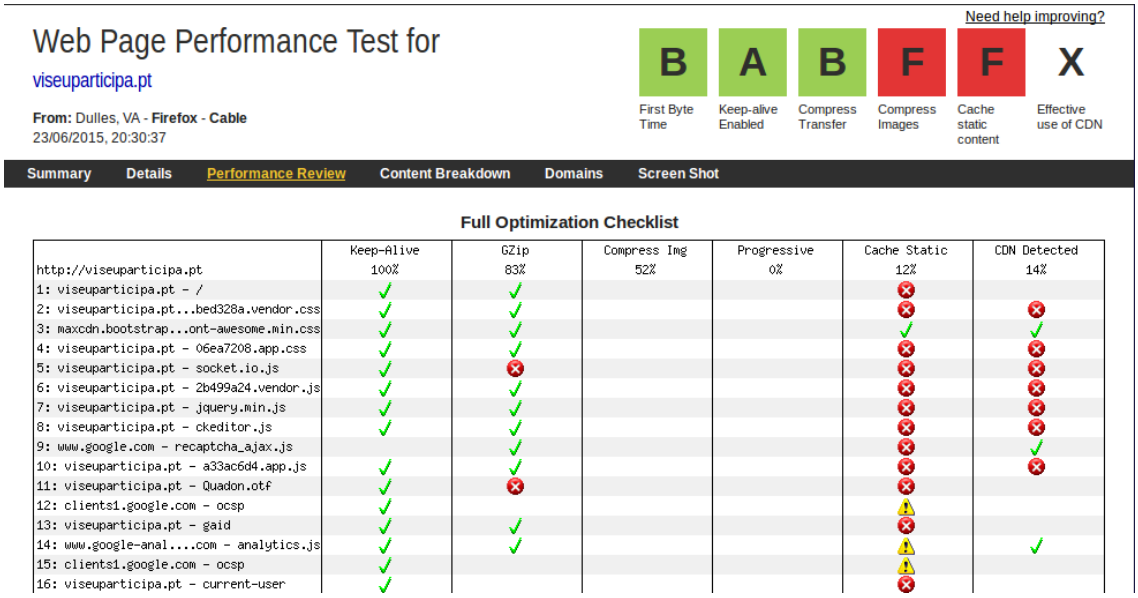


Figura 23: Formulário de submissão de testes no WebPagetest

Na Figura 23, está representado o formulário de seleção das condições de execução do teste. As configurações disponíveis são a escolha da localização, o *browser*, o tipo de ligação, o número de repetições do teste e o estado da *cache* no acesso à página escolhida para realizar o teste. Por defeito, as configurações avançadas utilizadas no teste são uma conexão por cabo de 5 Mbps de *download* e 1 de *upload* e a realização de dois testes, um com a *cache* vazia e outro com ela preenchida.

A realização dos testes pode ser feita com recurso a um conjunto de scripts e API RESTful disponibilizados pelo autor da plataforma. A sua utilização permite que a execução e recolha de dados sejam agendadas programaticamente. A API restringe o número de testes a 200 carregamentos de página por dia. Um teste em que se apresentem os resultados nos cenários em que a *cache* está preenchida e vazia, conta como dois carregamentos de página. A limitação de carregamentos pode ser ultrapassada, com a criação de uma instância privada WebPagetest (https://github.com/WPO-Foundation/webpagetest). A utilização de uma instância privada tem a vantagem de utilizar uma máquina privada para realizar os testes. Por vezes a quantidade de testes pedidos na plataforma pública, por todos os utilizadores, obriga a que haja uma espera maior para a recolha de dados.

Os resultados do teste são apresentados em várias secções. À semelhança de outras ferramentas apresentadas, esta plataforma também classifica e dá recomendações. Após o teste executado, os resultados podem ser vistos na plataforma. Na página "Performance Review", visualizam-se as recomendações de melhorias, como ilustra a Figura 24.



Web Page Performance Test for
viseuparticipa.pt
From: Dulles, VA - Firefox - Cable
23/06/2015, 20:30:37

Need help improving?

B	A	B	F	F	X
First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content	Effective use of CDN

Summary | Details | Performance Review | Content Breakdown | Domains | Screen Shot

Full Optimization Checklist

	Keep-Alive	GZip	Compress Img	Progressive	Cache Static	CDN Detected
http://viseuparticipa.pt	100%	83%	52%	0%	12%	14%
1: viseuparticipa.pt - /	✓	✓			✗	
2: viseuparticipa.pt...bed328a.vendor.css	✓	✓			✗	✗
3: maxcdn.bootstrap...ont-awesome.min.css	✓	✓			✓	✓
4: viseuparticipa.pt - 06ea7208.app.css	✓	✓			✗	✗
5: viseuparticipa.pt - socket.io.js	✓	✗			✗	✗
6: viseuparticipa.pt - 2b499a24.vendor.js	✓	✓			✗	✗
7: viseuparticipa.pt - jquery.min.js	✓	✓			✗	✗
8: viseuparticipa.pt - ckeditor.js	✓	✓			✗	✗
9: www.google.com - recaptcha_ajax.js		✓			✗	✓
10: viseuparticipa.pt - a33ac6d4.app.js	✓	✓			✗	✗
11: viseuparticipa.pt - Quadon.otf	✓	✗			✗	
12: clients1.google.com - ocsp	✓				⚠	
13: viseuparticipa.pt - gaid	✓	✓			✗	
14: www.google-anal....com - analytics.js	✓	✓			⚠	✓
15: clients1.google.com - ocsp	✓				⚠	
16: viseuparticipa.pt - current-user	✓				✗	

Figura 24: Recomendações do WebPagetest

Os recursos são avaliados um a um pelos seguintes aspetos de avaliação, quando se aplicam.

- O tempo de resposta do servidor depois da conexão TCP estabelecida (First Byte Time);
- A reutilização das conexões TCP para fazer pedidos de rede ao mesmo domínio (Keep-alive Enabled);
- A compressão dos recursos, recorrendo a programas que utilizam algoritmos para o efeito, como por exemplo, o Gzip;
- A compressão de imagens;
- As políticas de armazenamento de recursos estáticos em *cache* como imagens, ficheiros CSS e javascript;
- A utilização de um CDN, para fornecer melhores tempos a utilizadores dispersos geograficamente.

A página "Details" apresenta o gráfico em cascata, representado na Figura 25, com todos os recursos transferidos e o tempo da sua transferência. Este gráfico também está presente no Chrome DevTools. Ele indica quando o conteúdo da página está pronto para ser renderizado (*Dom Content Loaded*), o início da renderização e quando a renderização está completa. Verifica-se na Figura 25 o momento em que é iniciada a renderização (linha vertical verde perto dos 2 segundos) e quando a página está completa visualmente (linha vertical azul perto dos 7 segundos e meio).

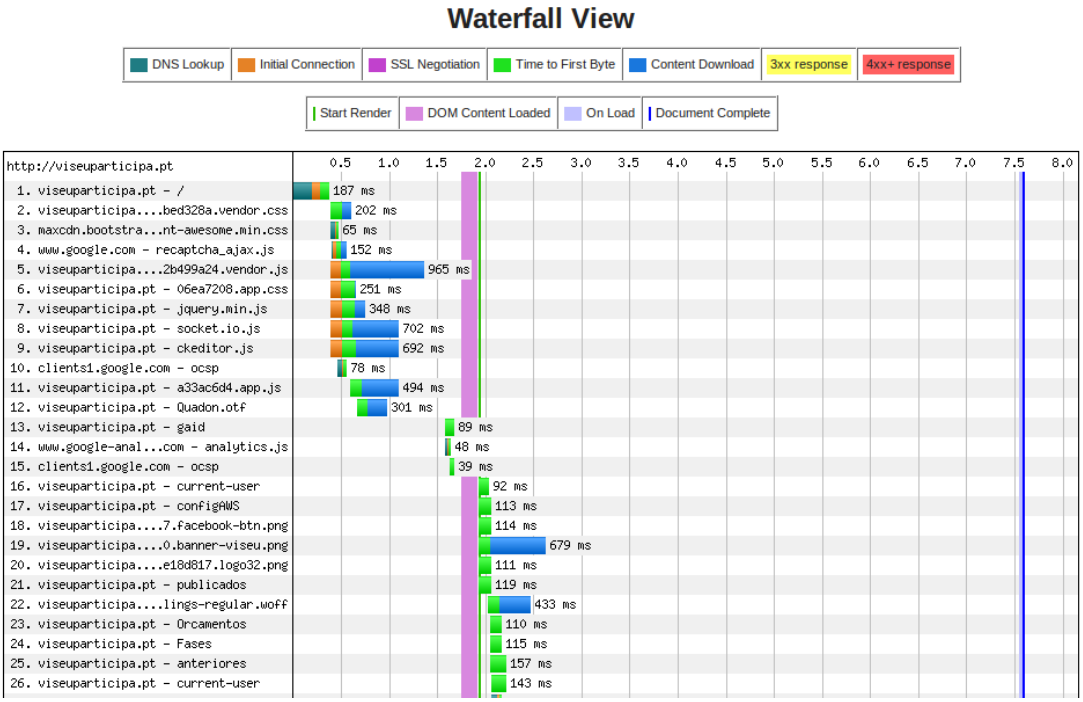


Figura 25: Gráfico em cascata no WebPagetest

A Figura 26 representa uma tabela com informação dos valores das métricas no carregamento da página. Apresentam-se os valores das métricas na primeira visualização de página e na segunda, onde já há recursos em *cache*.

						Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View	7.589s	0.366s	1.929s	2948	505	7.589s	72	8,175 KB	8.144s	76	8,205 KB	$$$$$
Repeat View	4.063s	0.248s	1.506s	2405	501	4.063s	50	844 KB	4.228s	51	850 KB	

Figura 26: Tabela de tempos no WebPagetest

A tabela evidencia os valores das métricas de performance em dois momentos diferentes do carregamento da página testada. O "Document Complete" ocorre quando todos os recursos da página Web acabam de ser transferidos. Após a transferência dos vários recursos, entre eles podem estar recursos javascript, a partir dos quais se executam pedidos assíncronos de outros recursos. O autor da plataforma define o "Fully Loaded" como o momento em que cessam as solicitações de pedidos de rede durante o período de 2 segundos.

O *Speed Index* mede o progresso visual da página e indica o tempo que ela demora a ser renderizada completamente. O método utilizado pelo WebPagetest consiste em capturar fotografias em diferentes momentos do carregamento da página para avaliar o estado visual da página Web. A cada segundo são recolhidas 10 imagens, também denominadas *frames*. Atribui-se a cada *frame* a percentagem da visualização completa da página, como representado na Figura 27.

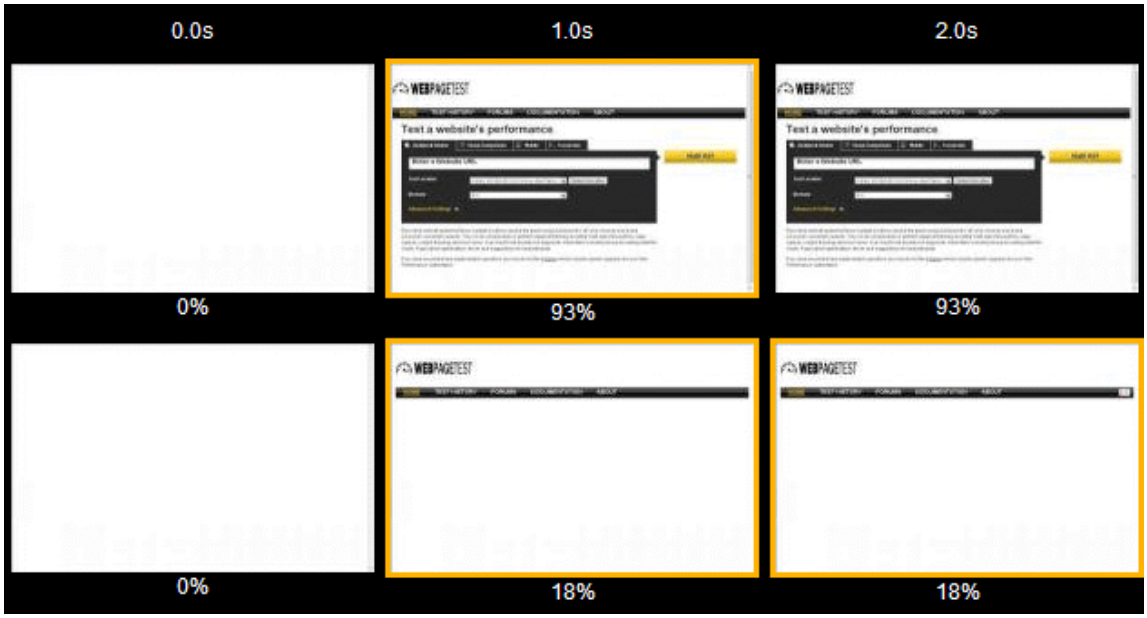


Figura 27: Filme de amostragem de uma página no decorrer do tempo [20]

4.2.2 SpeedCurve

O SpeedCurve (https://speedcurve.com) é uma plataforma que disponibiliza um serviço de análise de performance pago. Os dados recolhidos passam pela execução de testes sintéticos utilizando instâncias privadas do projeto WebPagetest. A plataforma satisfaz algumas lacunas da plataforma *online* WebPagetest, como por exemplo, a representação da evolução das várias métricas ao longo do tempo.

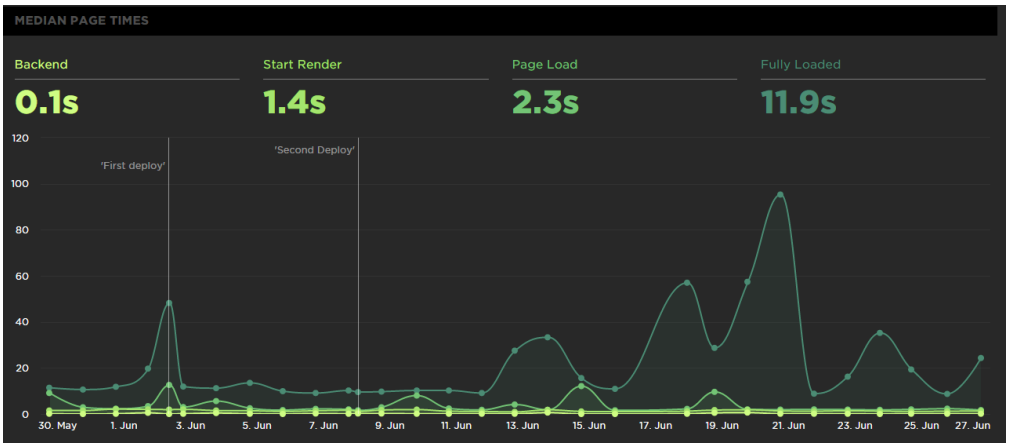


Figura 28: Evolução temporal da média das métricas

Na Figura 28 está representada a evolução temporal da média do valor das métricas do tempo passado no *Backend*, no início de renderização, no carregamento de página e no carregamento completo da página. As barras verticais que contêm as legendas "First deploy" e "Second deploy", correspondem aos momentos em que houve atualizações da aplicação Web. Através do gráfico mede-se o impacto das alterações das versões nos valores das métricas de performance. Na Figura 29 comparam-se os valores das métricas de performance nos dois dias em que a plataforma Web foi atualizada.

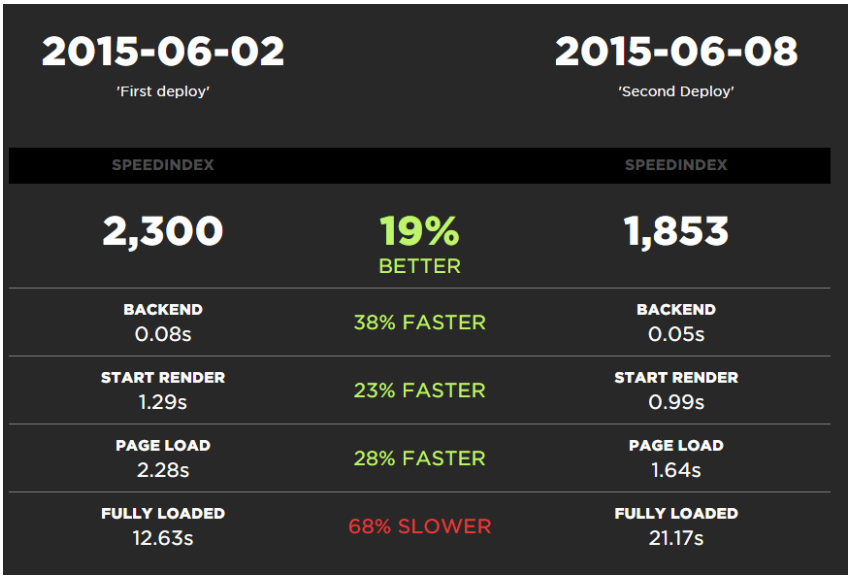


Figura 29: Comparação entre duas versões de *software*

O *Speed Index* é uma métrica que depende do tamanho do dispositivo que acede às páginas Web. A variedade de diferentes tamanhos de ecrãs, torna difícil a análise da experiência de utilização em cada um deles. No SpeedCurve é possível visualizar a captura de vídeo, funcionalidade suportada pelo WebPagetest, em ecrãs de várias dimensões, como representa a Figura 30.

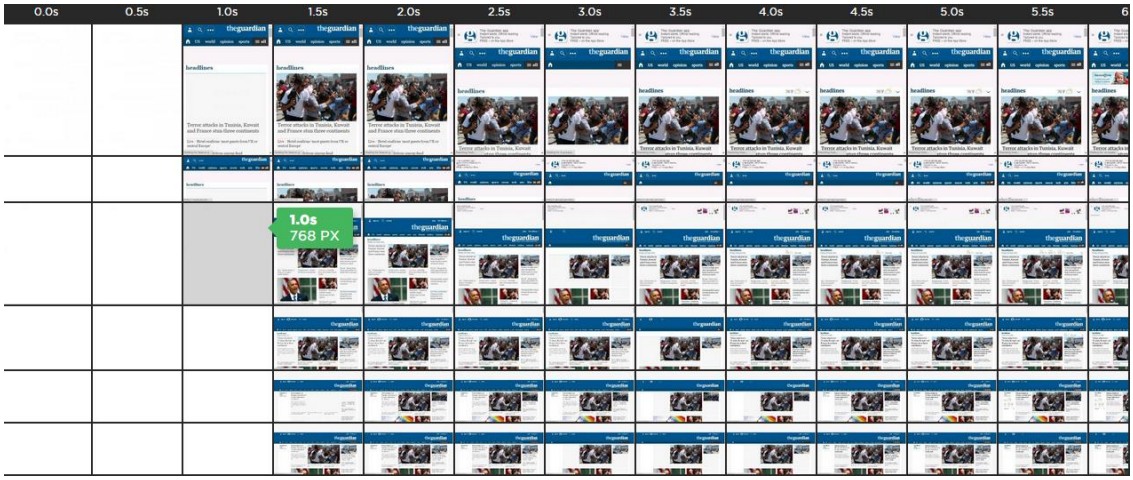


Figura 30: Evolução de uma página em dispositivos com diferentes tamanhos

A Figura 30 tem várias linhas e cada uma delas corresponde à experiência de utilização de um dispositivo com um tamanho diferente de ecrã. As capturas de vídeo estão ordenadas por ordem crescente de tamanho de ecrã, de cima para baixo. A terceira linha corresponde aos dispositivos cujos ecrãs têm 768px de largura. Verifica-se que na página testada, o início da renderização é mais rápido nos dispositivos com ecrãs menores.

A competitividade *online* entre empresas que exploram o mesmo nicho de mercado obriga a que haja uma necessidade de comparar medidas de performance das suas aplicações Web.

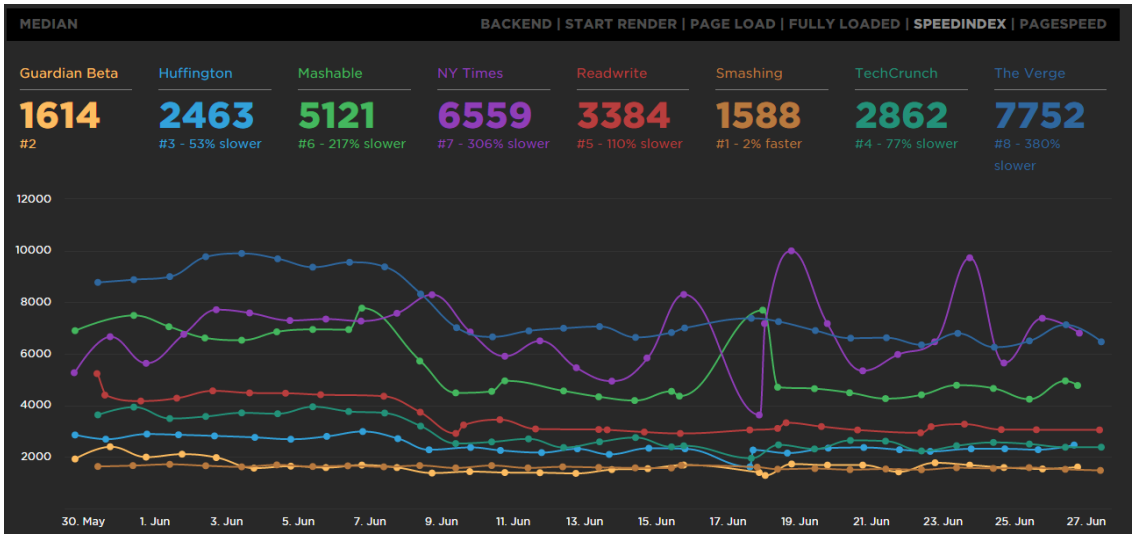


Figura 31: *Benchmark* de *Speed Index*

A Figura 31 ilustra um gráfico onde é analisada a evolução do valor da métrica *Speed Index* em páginas de oito aplicações Web.

A performance necessita de ser monitorizada. É importante estabelecer os limites que aos valores das métricas de performance não devem ultrapassar. Mantendo os valores abaixo dos limites estabelecidos garante-se que a aplicação Web fornece a experiência de utilização pretendida.

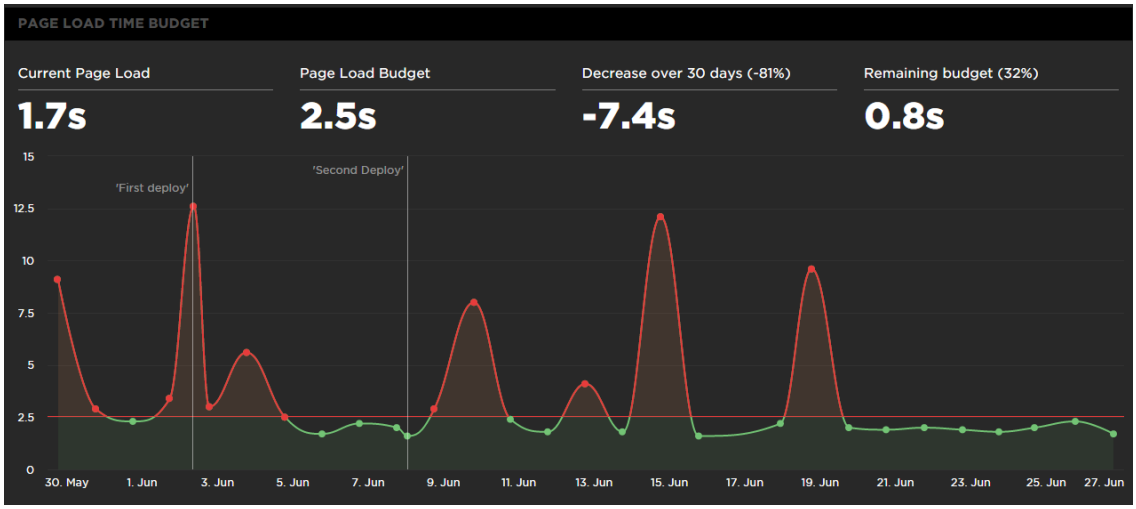


Figura 32: Controlo do valor de carregamento da página

Na Figura 32, visualiza-se o limite de tempo de carregamento estabelecido com o valor de 2 segundos e meio. Verifica-se que o limite é ultrapassado várias vezes. A partir do dia 20 de Junho, o valor da métrica mantem-se abaixo do limite.

Segundo os criadores do Speedcurve (https://speedcurve.com/), o conteúdo de terceiros é responsável por 40 a 60% do tamanho transferido numa página Web. Entre estes estão anúncios publicitários e scripts de monitorização. Este conteúdo, por vezes, não é transferido assincronamente, o que pode bloquear a transferência dos outros recursos e aumentar o tempo de carregamento da página. A Figura 33 ilustra o impacto dos vários domínios no carregamento de uma página Web.

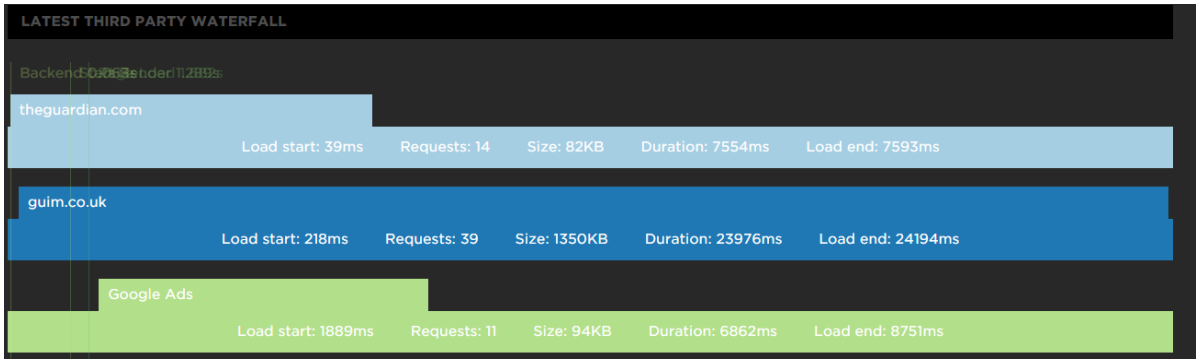


Figura 33: Valores das métricas de conteúdo externo

4.2.3 Phantomjs

O Phantomjs (http://phantomjs.org) é um meio para fabricar uma ferramenta de análise de performance. A utilização de plataformas *online* limita a análise da performance às funcionalidades disponibilizadas por elas.

O Phantomjs é um *headless browser*, um *browser* sem interface gráfica [30]. Ele disponibiliza um ambiente similar aos *browsers* normais mas é executado a partir de uma linha de comandos. Os *headless browsers* são úteis para testar a qualidade e interação das páginas Web, tirar fotografias do conteúdo renderizado, interpretar o HTML e analisar a performance.

A utilização desta ferramenta consiste na escrita de scripts com recurso à linguagem de programação javascript. Ela disponibiliza uma API para aceder ao webKit, que contém módulos de navegação na web, manipulação de elementos DOM e CSS e captura de imagens.

```
// Simple Javascript example

console.log('Loading a web page');
var page = require('webpage').create();
var url = 'http://phantomjs.org/';
page.open(url, function (status) {
  //Page is loaded!
  phantom.exit();
});
```

Figura 34: Script de phantomjs

O código presente na Figura 34 é um script Phantomjs que realiza um acesso à página http://phantomjs.org. Existem alguns projetos que utilizam o Phantomjs para executar testes de performance.

A análise de performance feita pelo Yslow pode ser feita a partir de um script em Phantomjs (http://yslow.org/phantomjs). Ele analisa a performance de uma página Web e gera ficheiros para serem utilizados em *frameworks* de automação de testes. As ferramentas de automação de testes podem ser utilizados para controlar os valores de desempenho das aplicações.

```
Fabio@XYZ ~/www/yslow-phantomjs-3.1.8
$ phantomjs yslow.js --info basic --format plain http://covilhadecide.pt
version: 3.1.8
size: 8355.2K (8355269 bytes)
overall score: C (73)
url: http://covilhadecide.pt/
# of requests: 36
ruleset: ydefault
page load time: 9423
```

Figura 35: Análise de performance com o projeto yslow em Phantomjs

Os resultados do teste executado à página http://covilhadecide.pt, presentes na Figura 35, indicam que foram feitos 36 pedidos de rede, foram transferidos 8355 Kilobytes, o tempo de carregamento foi 9.4 segundos e a classificação geral é "C". Os parâmetros avaliados são os apresentados na secção do YSlow.

O script do projeto YSlow executa testes de performance, cujos resultados podem ser utilizados em *frameworks* de automação de testes. Na execução do comando que realiza o teste sintético definem-se argumentos que, por exemplo, detetam todos os aspetos de avaliação que têm uma classificação inferior a "B". Os resultados de um teste sintético executado são apresentados na Figura 36.

```
Fabio@XYZ ~/www/yslow-phantomjs-3.1.8
$ phantomjs yslow.js --info grade --format tap --threshold '{"overall":"B"}' ht
tp://viseuparticipa.pt
TAP version 13
1..24
not ok 1 C (72) overall score
not ok 2 F (27) ynumreq: Make fewer HTTP requests
  ---
  message: This page has 15 external Javascript scripts.  Try combining them int
o one.
This page has 3 external stylesheets.  Try combining them into one.
This page has 13 external background images.  Try combining them with CSS sprite
s.
  ...
not ok 3 F (0) ycdn: Use a Content Delivery Network (CDN)
  ---
```

Figura 36: Teste de análise de performance com o projeto yslow em Phantomjs

Os resultados de teste são utilizados como *input* de uma *framework* de automação de testes. Por exemplo, nos resultados do comando executado na Figura 36, são procurados todos os aspetos de avaliação com classificação abaixo de "B". A *framework* de automação de testes revelaria que existem três testes reprovados:

- A classificação geral é "C";
- O facto de haver grande número de pedidos HTTP é atribuído ao aspeto avaliado a classificação "F";
- O facto de não se utilizar de CDN classifica este aspeto como "F".

O projeto Confess, disponível em https://github.com/jamesgpearce/confess, tem um script que analisa a performance, cria uma *appcache*, que é uma versão *offline* de uma aplicação Web [31], e gera uma lista das propriedades CSS utilizadas na página Web.

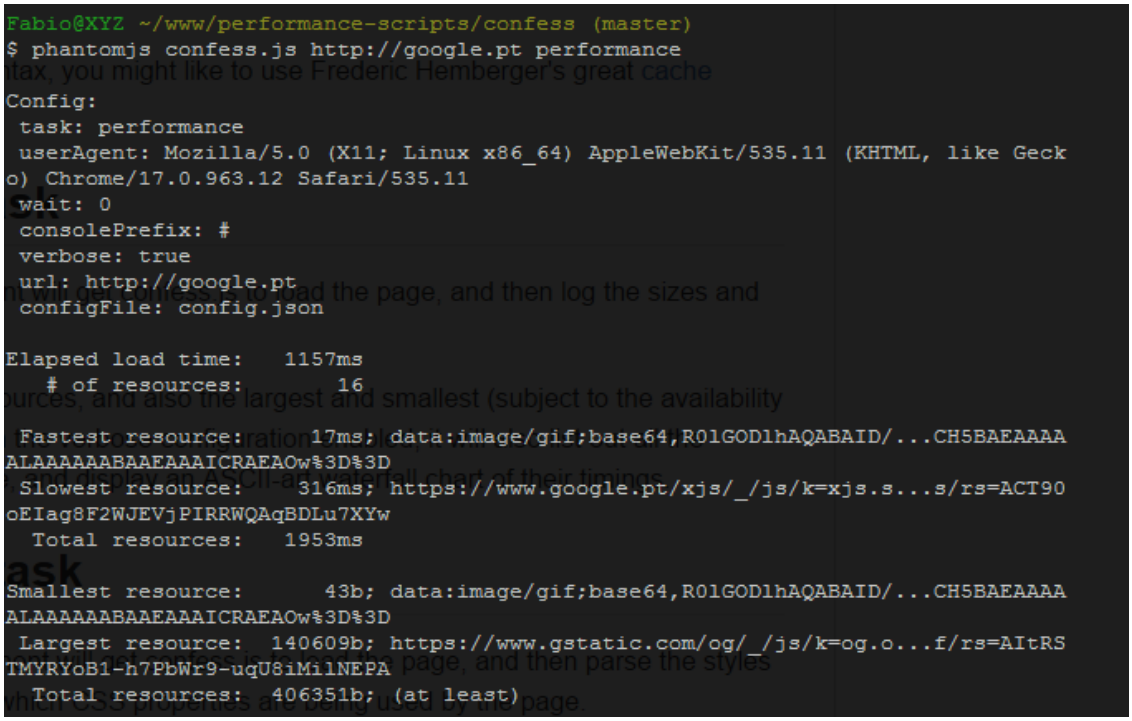


Figura 37: Informações de análise de performance do Confess

Os resultados da execução do comando do projeto Confess, representado na Figura 37, têm os valores de performance da página Web http://google.pt. Neles está presente o tempo de carregamento, o recurso maior, o menor, o mais rápido e o mais lento a ser transferido. Ainda se pode visualizar um gráfico em cascata, apresentado na Figura 38, com os recursos envolvidos no carregamento da página Web.

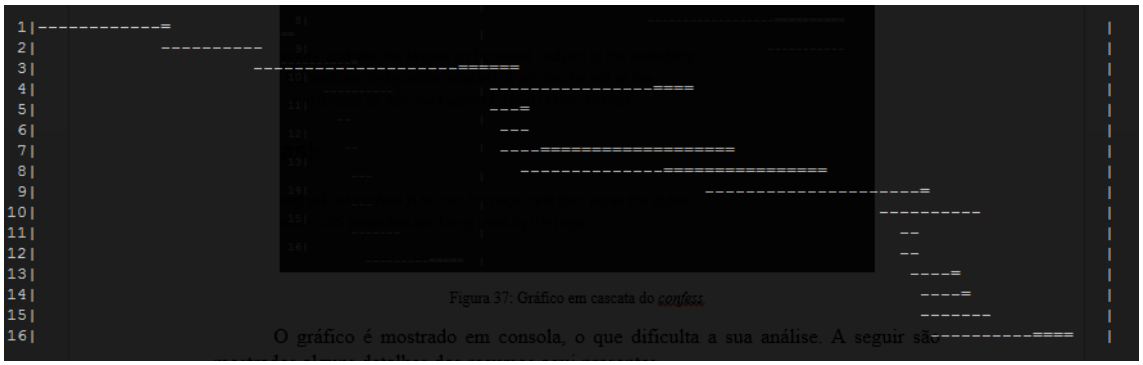


Figura 38: Gráfico em cascata do *confess*

O gráfico é visualizado em consola, o que dificulta a sua análise. Na Figura 39 estão representados detalhes dos recursos transferidos.

```
 1:    128ms;     218b; http://google.pt/
 2:    100ms;     230b; http://www.google.pt/
 3:    286ms;   55258b; https://www.google.pt/?gws_rd=ssl
 4:    215ms;    2019b; https://ssl.gstatic.com/gb/images/i1_0430b5ba.png
 5:     33ms;     219b; ...5TLMahGZ1H1EYAEcZakm46yD1AAAAAElFTkSuQmCC
 6:     29ms;    2706b; https://www.google.pt/logos/doodles/2015...zons-pluto-flyby-5641113681526784-res.png
 7:    242ms;     837b; https://www.google.pt/logos/doodles/2015...ons-pluto-flyby-5641113681526784.2-hp.gif
 8:    317ms;   52487b; https://www.google.pt/xjs/_/js/k=xjs.s.p...cms/rs=ACT90oEIag8F2WJEVjPIRRWQAqBDLu7XYw
 9:    226ms;  140609b; https://www.gstatic.com/og/_/js/k=og.og2...def/rs=AItRSTMYRYoB1-h7PbWr9-uqU8iMilNEPA
10:    102ms;   46005b; https://www.google.pt/extern_chrome/cc89d2709853a1ea.js?bav=on.2,or.
11:     22ms;      76b; ...pZ+suQJyy0ocV3bbm33EcCArmiUYk1qxAUAAAOw==
12:     17ms;      43b; ...AACH5BAEAAAAALAAAAAABAAEAAAICRAEAOw%3D%3D
13:     39ms;   60842b; https://www.google.pt/xjs/_/js/k=xjs.s.p...cms/rs=ACT90oEIag8F2WJEVjPIRRWQAqBDLu7XYw
14:     35ms;     151b; https://www.gstatic.com/inputtools/images/tia.png
15:     76ms;       -b; https://www.google.pt/gen_204?atyp=i&ct=...i=8CW1VYXzHabU7AaNk4LYDA&zx=1436886512725
16:    149ms;   47682b; https://apis.google.com/_/scs/abc-static...24Ahcib0nCMs4uGYi9i5Toag/cb=gapi.loaded_0
```

Figura 39: Detalhes dos recursos transferidos no *confess*

São apresentados os vários recursos envolvidos na página Web acedida e alguns dos seus detalhes, como tempo de *download*, tamanho em bytes e a sua fonte.

O projeto loadreport, disponível em https://github.com/wesleyhales/loadreport, permite executar testes de performance a um website e armazenar os resultados diretamente num ficheiro com formato CSV ou JSON. O documento CSV que armazena dados de todos os testes executados com o script tem as seguinte informações:

- O número de recursos;
- O tempo passado na transferência dos recursos;
- Os tempos dos eventos passados no *frontend* (domReadystateLoading, domReadystateInteractive, windowOnLoad);
- Os recursos que demoraram mais e menos a ser transferidos;
- Os recursos maior e menor;
- A data e hora a que foi executado o teste.

O mesmo script, utilizando argumentos diferentes, grava imagens do estado visual da página no decorrer do carregamento da página.

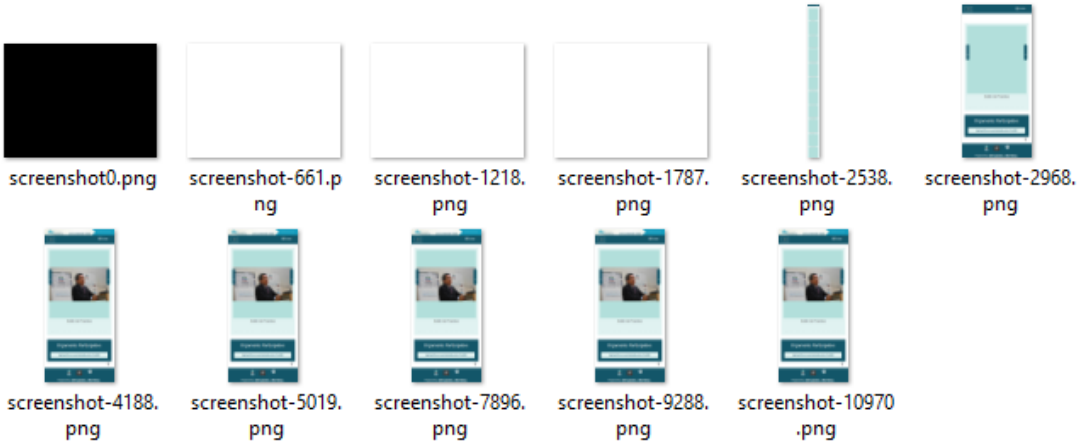


Figura 40: *Filmstrip* do *loadreport*

O script gera imagens, como ilustra a Figura 40, com o estado visual da página após determinado tempo. Os seus nomes indicam o número de milissegundos depois do início da navegação em que foi tirado o *snapshot*. No teste executado a renderização começa aos 2.5 segundos após o início da navegação e termina aos 10.9 segundos.

4.3 Ferramentas que monitorização utilizadores reais

Os testes sintéticos são executados em ambientes controlados, onde as condições dos testes são sempre as mesmas. A estabilidade dos dados recolhidos, ao longo do tempo, permite avaliar globalmente o desempenho das páginas Web e identificar o impacto das alterações de código.

Os resultados de performance analisados nos testes sintéticos não espelham os valores verdadeiros dos utilizadores. Steve Souders afirma que, tipicamente, os tempos de carregamento dos utilizadores reais são duas vezes superiores aos dos resultados dos testes sintéticos [28]. A diversidade de ambientes possíveis torna impossível a sua replicação de todos os cenários em testes sintéticos. Os utilizadores acedem à aplicação com *browsers* diferentes, estão em locais geográficos dispares, podem utilizar redes móveis ou cabladas, têm recursos ou não armazenados em *cache*, etc. Apresentam-se nesta secção algumas soluções de medição de desempenho do acesso às páginas Web por utilizadores.

4.3.1 Soasta mPulse

A Soasta é uma empresa norte americana que fornece serviços para analisar a performance e testar as aplicações web (http://www.soasta.com). Um dos seus produtos é o mPulse, centrado na análise da performance de utilizadores reais. O mPulse é um serviço pago, mas pode ser utilizado gratuitamente, tendo a limitação dos dados de performance serem guardados apenas durante o período de um mês. A seguir são apresentados alguns elementos gráficos disponibilizados na plataforma Web mPulse para analisar a performance dos utilizadores reais.

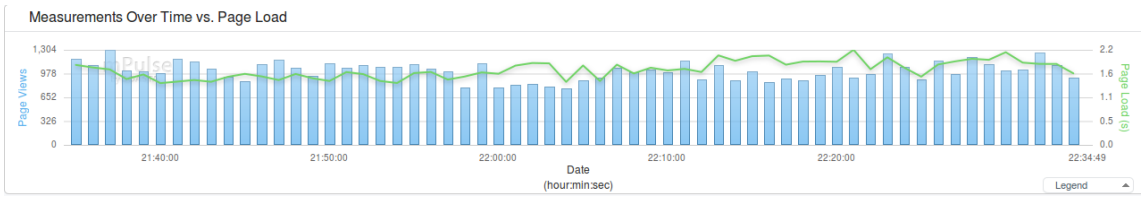


Figura 41: Tempo de carregamento e número de páginas visualizadas no mPulse

A Figura 41 ilustra um gráfico onde são apresentados os resultados do tempo de carregamento e o número de páginas visualizadas por minuto. As barras identificam o número de visualizações e a linha indica o valor médio do tempo de carregamento da página, em segundos.

A diversidade de condições às quais estão sujeitos os utilizadores levam a que o tempo médio de carregamento possa variar de dia para dia, chegando mesmo a ter valores extremamente díspares. Interessa analisar o desempenho das aplicações tendo em conta as condições de acesso dos utilizadores, de forma a tentar identificar algum aspeto que deva ser corrigido.

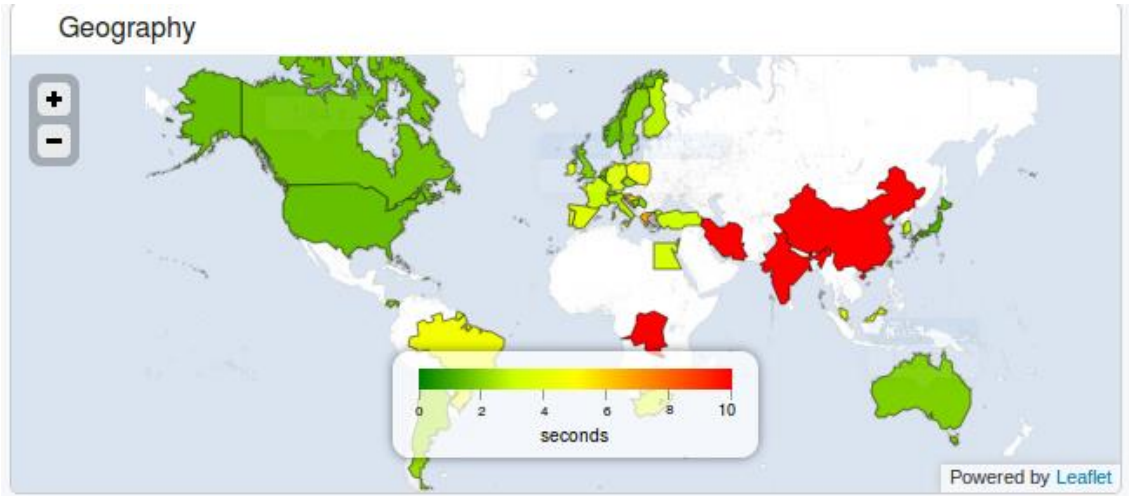


Figura 42: Tempo de carregamento de página por área geográfica

Na Figura 42 analisa-se o tempo médio de carregamento em diferentes países. Verifica-se que a aplicação Web em causa tem tempos de carregamento na ordem dos 10 segundos ou mais, em países como a Índia e China. Uma tentativa de reduzir os tempos de carregamentos nos países indicados pode passar por utilizar um servidor próximo dos utilizadores para servir os recursos da aplicação, isto é, utilizar um CDN.

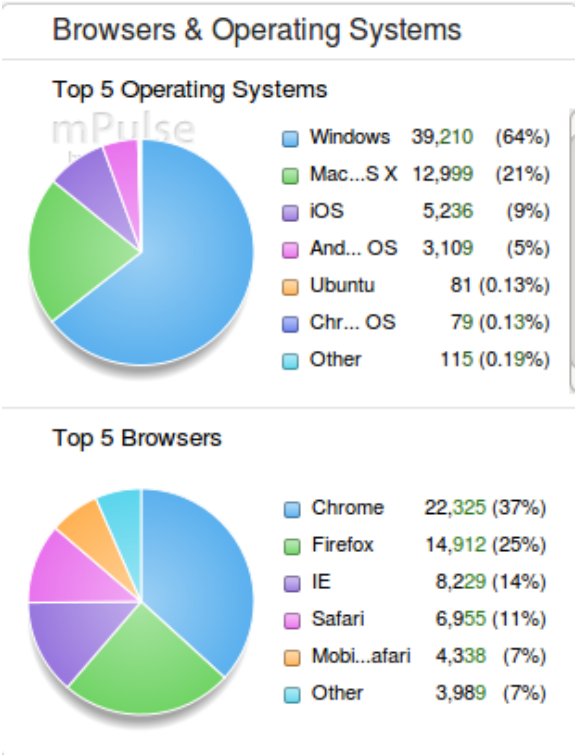


Figura 43: Percentagem de utilizadores que utilizam um *browser* e sistema operativo

Os gráficos de setores apresentados na Figura 43 evidenciam o contributo da utilização de um sistema operativo e *browser* para os valores de performance de uma página Web. A plataforma permite filtrar os dados por país, sistema operativo, *browser*, tipo de dispositivos, tipo de tecnologia de rede, etc.

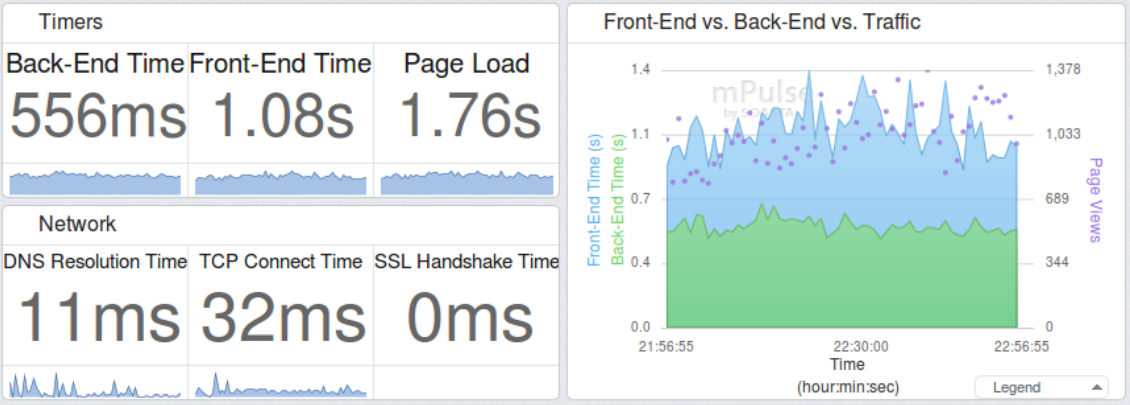


Figura 44: Frontend vs Backend (mPulse)

O *dashboard*, representado na Figura 44, indica o tempo médio passado no *backend*, *frontend* e o tempo de carregamento. O tempo passado no *backend* inclui o tempo de pesquisa DNS, conexão TCP e negociação SSL, cujos tempos médios também estão representados na Figura 44. No gráfico visualiza-se o impacto do tempo passado no *backend* e *frontend*, no tempo de carregamento. Notoriamente, o *frontend* tem maior impacto. Este tipo de informações permite à equipa de desenvolvimento da aplicação, detetar onde se deve procurar melhorar a performance da aplicação. No *backend* procuram-se melhorar os processos que ocorrem no servidor, como por exemplo, as operações na base de dados, o processamento no servidor, configurações da máquina ou protocolos de conexão. No *frontend* analisam-se as operações que ocorrem depois do utilizador receber a página HTML, como o número de recursos presentes na página, a ordem de sua chamada, etc.

4.3.2 Google Analytics

O Google Analytics é uma plataforma desenvolvida pela Google, com o propósito de fornecer informação sobre estatísticas de utilização de um *website*. Ela fornece relatórios com o número de visitas, páginas visualizadas, tempo médio permanecido na aplicação, taxa de utilizadores que saem da aplicação após visualizar a primeira página, dados de performance e outras métricas. A plataforma revela-se útil na análise do impacto da melhoria de performance da aplicação, verificando-se por exemplo, se o número de visitantes aumenta com a diminuição do tempo de carregamento de página.

A recolha de dados sobre a performance, por parte do Google Analytics, necessita da alteração das configurações de recolha de dados. O script de recolha de dados está configurado para extrair informação de performance em apenas 1% das páginas visualizadas pelos utilizadores. Uma recolha mais intensiva destes dados consiste em alterar essas configurações. A seguir são apresentadas as secções da plataforma que apresentam os dados de desempenho recolhidos.

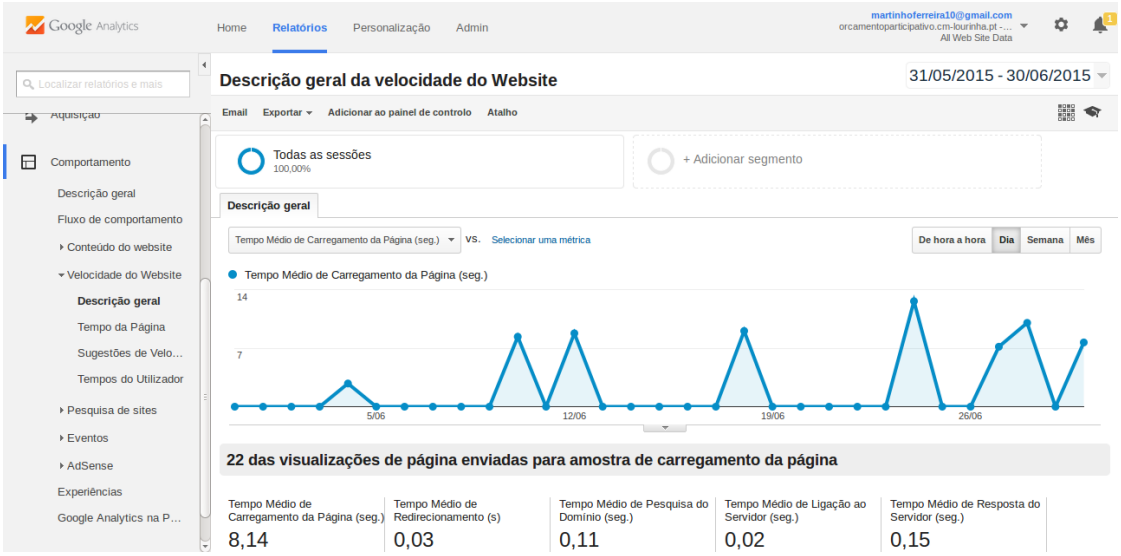


Figura 45: Velocidade do Website no Google Analytics

Os dados sobre os tempos de carregamento estão na secção "Comportamento", dentro do separador "Velocidade do *website*". A Figura 45 ilustra a página que apresenta dos resultados gerais da performance de um *website.*

Os relatórios de velocidade avaliam os tempos de espera em dois tipos de eventos:

- **O tempo de carregamento de página**. O tempo pode ser examinado por tipo de navegador, área geográfica e página. Os dados sobre estes eventos são visualizados ao clicar na coluna lateral esquerda em "Tempos de Página";
- **O tempos de eventos personalizados pelos utilizadores**. O programador utiliza uma API disponibilizada pela plataforma, para programar o início e fim de um evento, cuja duração é enviada para a plataforma. Os eventos monitorizados podem ser, por exemplo, o carregamento de uma imagem ou o tempo de interação após o clique num botão. A informação sobre os eventos programados está em "Tempos do Utilizador".

Ambas as páginas referidas para os dois tipos de eventos têm três secções que permitem visualizar os dados em perspetivas diferentes. As áreas são o "Explorador", "Distribuição" e "Cobertura geográfica".

No "Explorador" visualiza-se a evolução dos valores das várias métricas ao longo do tempo. Um gráfico presente nessa página permite comparar os valores de duas métricas. As métricas são divididas em três categorias, descritas a seguir.

- **Utilização do Website** - contém métricas sobre a utilização do *website*, como a taxa de rejeição e tempos médios de carregamento de página;
- **Dados Técnicos** – indica os valores de tempo relativos ao *backend*, como os tempos de estabelecimento da conexão TCP, da pesquisa DNS, de resposta do servidor e de transferência;
- **Tempos do DOM** – fornece os tempos de eventos que ocorrem no *frontend*, as métricas deste conjunto são o tempo médio de carregamento do conteúdo do documento e o tempo médio para início da interação com a página.

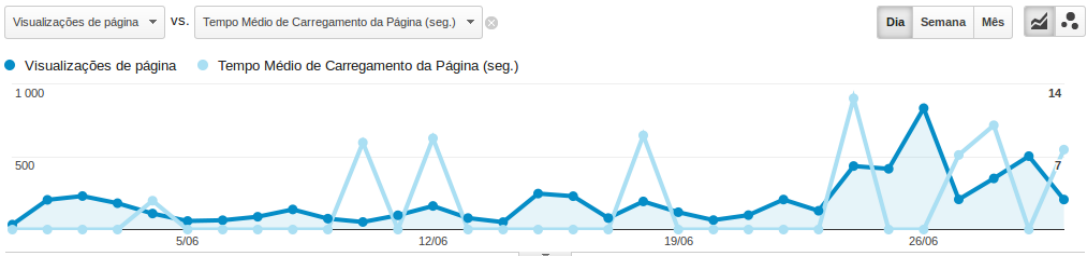


Figura 46: Tempo médio de carregamento de página e número de visualizações

Na Figura 46 comparam-se os tempos médios de carregamento da página com o número de visualizações de página. Os dados visualizados no gráfico podem ainda ser filtrados por página, tecnologias (*browser*, sistema operativo, resolução de ecrã, etc) e área geográfica.

Outro aspeto analisado nesta página é a comparação do valor de várias métricas entre grupos de utilizadores com determinadas características de acesso. Verifica-se a influência de cada um dos grupos no valor global da métrica.

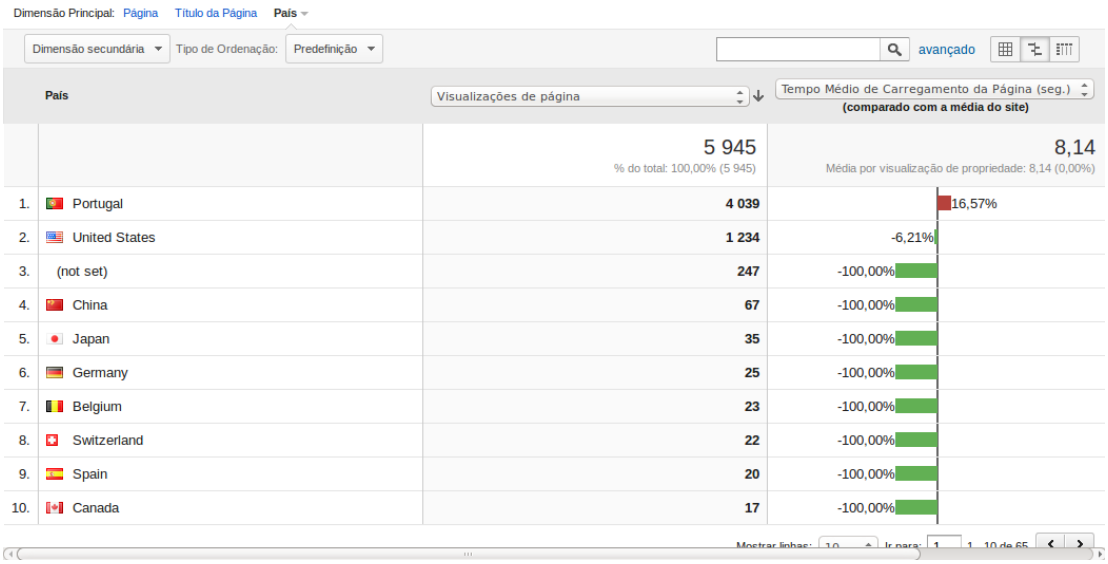


Figura 47: Influência de grupos no valor global de uma métrica

Por exemplo, na Figura 47, verifica-se que um grupo de utilizadores que acede a partir de Portugal tem um contributo negativo para a média do tempo de carregamento de página global, uma vez que o seu tempo médio é 16.57% superior. Por outro lado, utilizadores que acedem à aplicação através dos Estados Unidos contribuem para que o tempo médio de carregamento global seja mais baixo, o seu tempo médio é 6.21% menor ao tempo global. Neste caso os valores de comparação com o valor médio nos restantes países é indicado o valor -100%, porque não foram recolhidos dados sobre o desempenho apesar de terem ocorrido visitas. A informação fornecida pela Figura 47 pode incentivar a procurar soluções para melhorar o tempo médio de carregamentos dos utilizadores que acedem a partir de Portugal.

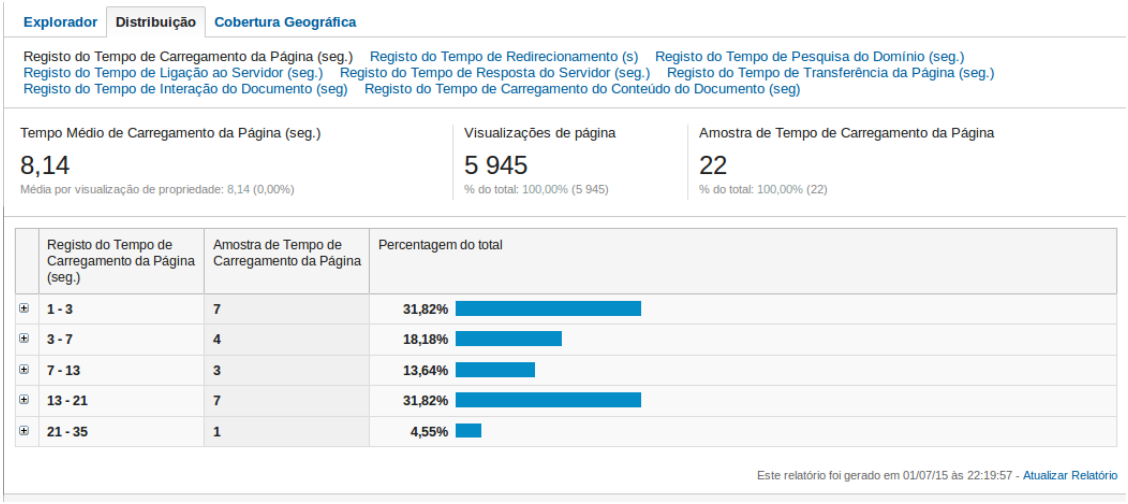


Figura 48: Percentagem de amostras por limites de tempo

Na Figura 48 está representada a página "Distribuição". Nela analisa-se o número de amostras por intervalo de valores de uma métrica. Por exemplo, na Figura 48, verifica-se que o tempo de carregamento de 4 visualizações de página variou entre 3 a 7 segundos. As métricas analisadas são todas expressas em segundos. Entre as métricas analisadas estão eventos que decorrem no *backend* e *frontend.* A informação das distribuições é acompanhada pela percentagem da amostragem total.

Na "Cobertura Geográfica", avalia-se o valor das várias métricas nos países a partir dos quais os utilizadores acederam ao *website*.

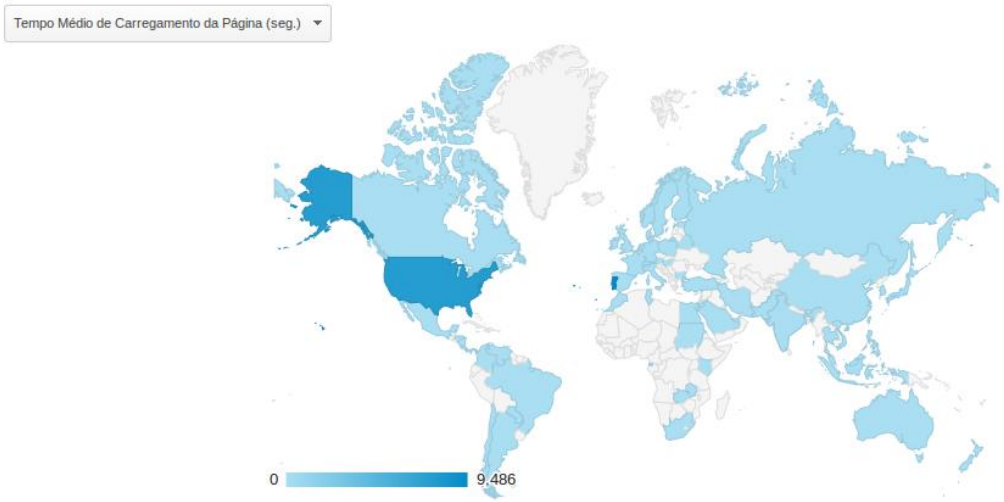


Figura 49: Tempo médio de carregamento por região geográfica

Na Figura 49, representam-se num mapa, o valor médio do tempo de carregamento em cada país. As zonas com a cor mais escura correspondem a tempos de carregamento maiores. A grande parte dos países tem uma cor mais clara, que indica que o valor médio do tempo de carregamento é zero segundos, isto porque, apesar de haver páginas visualizadas a partir desses países, não foram recolhidas informações sobre o desempenho. O tempo médio de carregamento é apenas uma das métricas que se pode avaliar.

Além dos relatórios presentes nas três áreas descritas, existe uma página do Google Analytics que fornece o número de visualizações, o tempo médio de carregamento de cada

página do *website* onde já foram recolhidos dados e recomendações de melhoria específicas para cada uma delas, como ilustrado na Figura 50.

	Página	Visualizações de página	Tempo Médio de Carregamento da Página (seg.)	Sugestões PageSpeed	Pontuação PageSpeed
1.	/home	1 747	3,23	6 no total	49
2.	/	840	12,67	6 no total	47
3.	/op/projetos	530	0,00	5 no total	67
4.	/op/home	391	0,00	6 no total	65
5.	/op/projetos#topLPropostas	221	0,00	5 no total	74
6.	/noticias/556c1a4326f4710b00f1b912	208	0,00	5 no total	68
7.	/op/propostas	169	0,00	6 no total	56
8.	/agenda/556c19df26f4710b00f1b911	159	0,00	5 no total	67
9.	/noticias/558a920038eaeb0d0071ace7	128	0,00	5 no total	67
10.	/op/projetos/55645bc58651bd0e006ec8aa	99	0,00	5 no total	67

1 - 10 de 208

Figura 50: Recomendações do Google Analytics

4.3.3 Javascript

O aparecimento do javascript na Web impulsionou a monitorização de utilizadores reais. Primeiro de uma forma mais complexa e menos precisa, utilizando o objeto Date para cronometrar os vários eventos de carregamento de página, como apresentado no código da Figura 51.

```
var startTime=new Date(),loadTime;

window.onload=function(){
    var endTime=new Date();
    loadTime=endTime-startTime;
    console.log('Load time is '+loadTime+' ms.');
}
```

Figura 51: Medição do tempo de navegação com objetos “Date”

A abordagem revela-se ineficiente porque as instruções de medição afetam o tempo de carregamento da página e apenas é possível contabilizar o tempo após receber o código javascript que realiza a medição, o que torna impossível fazer a medição do tempo passado em processos de rede como a pesquisa DNS e ligação TCP. A solução a esta abordagem surge com a *Navigation Timing API*, já apresentada anteriormente.

A medição com recurso a código javascript, consiste em incluir em todas as páginas da aplicação o código que irá transferir o ficheiro de monitorização. O script monitoriza a informação temporal dos vários eventos e envia-a para um servidor. No servidor essa informação é processada e armazenada numa base de dados para mais tarde ser analisada.

Há projetos *open source* como o Episodes (https://github.com/stevesouders/episodes), criado por Steve Souders, e o Boomerang (https://github.com/lognormal/boomerang), desenvolvido por uma equipa de programadores da Yahoo, que disponibilizam um script para recolher informação de performance dos utilizadores.

A medição do tempo de carregamento com o Episodes, entre duas páginas do mesmo *website* é feita da seguinte forma [13]:

1. Antes de sair da página, no evento *unload*, o script de monitorização armazena num *cookie* o momento em que o utilizador sai da página;
2. O script, na nova página, procura pelo cookie que tem o instante em que o utilizador saiu da página anterior;
3. A recolha de dados culmina no decurso do evento javascript "onLoad". Entretanto são recolhidos os tempos de vários eventos que contribuem para aumentar o tempo de carregamento, como por exemplo, os processos de rede e os eventos de *frontend.* A partir de uma API disponibilizada com o script é possível programar a monitorização de eventos, como por exemplo, o tempo de carregamento de uma imagem;
4. Por fim a informação recolhida é enviada para um servidor.

O projeto Boomerangjs disponibiliza funcionalidades mais básicas de medição de desempenho e *plugins* que permitem obter informação adicional. Os casos de uso e funcionalidades deste projeto são os seguintes:

1. Medir performance desde que o utilizador sai de uma página para outra, na mesma aplicação, à semelhança do Episodes;
2. Medir desempenho de recursos carregados dinamicamente;
3. Medir latência e largura de banda do utilizador;
4. Medir latência da procura DNS;
5. Medir aleatoriamente uma amostra dos utilizadores que acedem à aplicação, à semelhança do que pode ser feito com o parâmetro de configuração do Google Analytics;
6. Utilizar a *Navigation Timing Api* para enviar dados sobre o desempenho.

Dada a utilização de *plugins*, pode ser criado um script personalizado com as funcionalidades pretendidas.

5 Técnicas de melhoria de performance

A navegação na Web desencadeia processamento do lado do servidor e do cliente. Há várias obras literárias que ensinam estratégias para otimizar processos que ocorrem no servidor. Por exemplo, como devem ser configurados os serviços Web, criar índices na base de dados, etc. Steve Souders, na sua obra *High Performance Web Sites* [29], destaca a importância de melhorar os processos passados do lado do cliente, no *frontend*. Segundo a regra de ouro da performance, 80 a 90 por cento do tempo de espera do utilizador é passado no *frontend*, portanto a otimização de aplicações Web deve começar por aí [32]. A regra é fundamentada com a análise dos dados extraídos dos 10 web sites americanos mais utilizados segundo o ranking Alexa (http://www.alexa.com), em 2007. A percentagem de tempo passado a transferir a página HTML é menor que 20%, quer com a *cache* vazia como povoada, com exceção do site da Google, quando a *cache* tem recursos, como apresenta a tabela da Figura 52.

	Empty cache	Primed cache
AOL	6%	14%
Amazon	18%	14%
CNN	19%	8%
eBay	2%	8%
Google	14%	36%
MSN	3%	5%
MySpace	4%	14%
Wikipedia	20%	12%
Yahoo!	5%	12%
YouTube	3%	5%

Figura 52: Percentagem de tempo passado na transferência de páginas HTML [29]

O tempo considerado para o *backend*, é o tempo desde o início da navegação até ser recebido o primeiro byte do servidor. Nas aplicações tradicionais, onde compete ao servidor gerar a página completa antes de a enviar ao cliente, são efetuados aí os acessos à base dados ou Web *services*, prolongando o tempo de espera no *backend.* No *frontend* é contabilizado o resto do tempo na renderização da página HTML e na execução de javascript. A transferência dos recursos presentes na página é considerada um processo do *frontend*, pois as técnicas de melhorias do *frontend* podem melhorar esse tempo.

Existe, portanto, mais potencial de melhoria da performance da aplicação Web se forem utilizadas técnicas de melhoria de performance no *frontend.* Segundo a regra de ouro da performance, ao diminuir o tempo passado no *frontend* para metade, pode-se melhorar 45% do tempo de carregamento, enquanto melhorar metade do tempo de espera no *backend* melhora até 10% do tempo de carregamento.

Segundo Steve Souders [29], as técnicas de melhoria no *frontend* são mais fáceis e rápidas de implementar do que no *backend.* As melhorias no *backend* por vezes levam à reestruturação da aplicação em causa, o que pode implicar semanas ou meses de desenvolvimento.

A seguir apresentam-se técnicas de otimização dos tempos do *frontend* que levam à redução do tamanho dos recursos transferidos, à redução do número de pedidos de rede e à redução de tempo de latência na transferência de recursos presentes na página HTML. Averigua-se também o impacto do posicionamento das referências dos recursos a serem transferidos. Por fim analisa-se a compatibilidade das técnicas de melhorias nas aplicações Web 2.0. Ao mesmo tempo que se apresentam as técnicas, são apresentados resultados de alguns testes de forma a evidenciar as possíveis melhorias na aplicação Web no caso da sua implementação.

5.1 Reduzir tamanho dos recursos

Nos últimos anos tem-se assistido ao crescimento do tamanho das aplicações Web. O HTTPArchive (http://httparchive.org) apresenta o histórico de informações sobre a constituição dos *websites* mais utilizados pelos utilizadores, segundo a classificação do ranking Alexa (http://www.alexa.com). Na Figura 53 é apresentada a evolução do número médio de pedidos e o tamanho dos recursos transferidos desde Julho de 2012 até Julho de 2015.

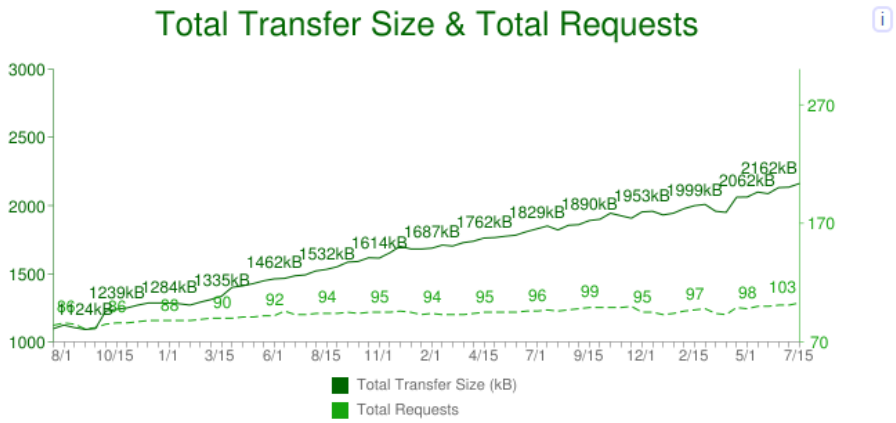


Figura 53: Tendência de crescimento do tamanho dos recursos transferidos

Nos últimos três anos, o tamanho médio dos *websites* analisados cresceram de 1124 Kilobytes para 2162 Kilobytes, quase para o dobro. As técnicas seguintes apresentadas pretendem contrariar esta tendência. Apesar das condições de largura de banda melhorarem de tempos, em tempos, não estão a acompanhar o crescimento das aplicações [1].

5.1.1 Imagens

As imagens são o recurso que contribui mais para aumentar o tamanho de uma página Web. Segundo dados extraídos a 15 de Julho de 2015, do HTTPArchive (http://httparchive.org), as imagens, em média, são responsáveis por mais de metade dos bytes do conteúdo transferido no acesso a uma página Web, como representa a Figura 54.

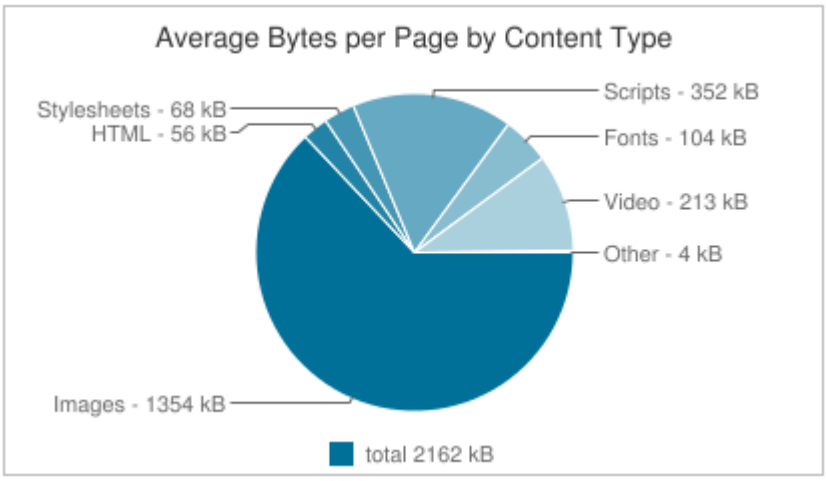


Figura 54: Tamanho médio do tipo de conteúdo por página

Devido ao seu tamanho e número, a forma mais fácil de diminuir o tamanho e tempo de resposta da página é otimizar as imagens. O processo de otimização começa pela escolha do formato correto, de acordo com as características das imagens. A seguir são apresentados os vários formatos de imagem existentes.

5.1.1.1 JPEG

O JPEG é o melhor formato para imagens com um largo espectro de cores. Este formato permite compressões, cujas alterações na imagem são impercetíveis ao olho humano. Quando reduzida a qualidade delas, podem notar-se alguns efeitos indesejados como áreas foscas e pixelização. Este efeito denomina-se *artifacting* [10]. A sua alteração em programas de edição de imagem, induzem a compressão de imagens JPEG, descartando informação dificilmente distinguível. A grande capacidade de redução de tamanho destas imagens, torna-o o formato preferido na Web. As imagens podem ser comprimidas comprometendo a sua qualidade. Na Figura 55 estão imagens exportadas com diferentes qualidades, ou seja, diferentes níveis de compressão.

Figura 55: Imagens JPEG exportadas com diferentes qualidades

Note-se que na imagem com a qualidade 25%, há mais pixelização. Isto é visível em zonas onde há maior contraste de cores.

Quanto mais cores a imagem possuir, maior será o tamanho dela e mais difícil será a compressão descartar informação não percetível. A Figura 56 apresenta um conjunto de imagens onde é introduzido propositadamente ruído.

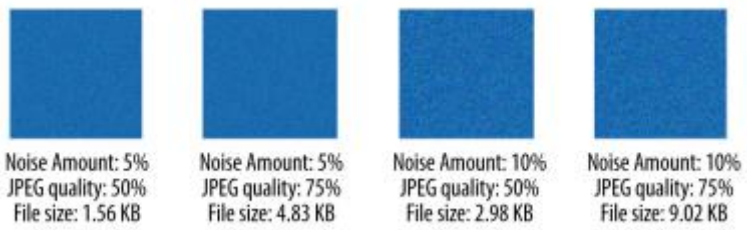


Figura 56: Imagens JPEG com diferentes níveis de ruído

O ruído introduzido aumenta o número de cores e consequentemente o tamanho das imagens. As imagens com ruido de 10% têm aproximadamente o dobro do tamanho das que têm um ruído de 5%.

Existem opções de criação de uma imagem JPEG que afetam a perceção da rapidez da sua renderização na página.

- **O "Baseline JPEG"** revela a imagem na página Web linha a linha, de cima para baixo.

- **O JPEG progressivo** é renderizado numa sequência de imagens, da que tem menor qualidade para a que tem mais, até ela estar completamente carregada.

O JPEG progressivo aparenta ser carregado mais rápido, uma vez que o espaço dele é ocupado todo de uma só vez, em vez de ser renderizado linha a linha como no "Baseline JPEG".

5.1.1.2 GIF

O GIF foi criado para guardar múltiplas imagens bitmaps, num único ficheiro, tornando possível a produção de animações. Suporta transparência e uma palete de até 256 cores e, ao contrário do JPEG, não perde qualidade. A sua utilização está em desuso e recomenda-se [10] utilizar apenas em duas situações. São elas as seguintes:

- O tamanho do ficheiro gerado no GIF é menor do que o tamanho da imagem como PNG-8.
- As animações não podem ser substituídas com CSS3.

O algoritmo de compressão do GIF remove as cores redundantes que se encontram na horizontal. Quer isto dizer que a imagem terá um tamanho maior, quantas mais cores possuir cada linha horizontal. O formato PNG-8 é a escolha certa para a maioria das imagens com poucas cores, pois para além de eliminar os padrões repetidos na horizontal, também o faz na vertical. Quanto às animações, deve-se considerar o uso de animações CSS3, elas tendem em ser mais leves e melhores em termos de performance [10]. Por isso, na maioria dos casos a substituição de um GIF é benéfica para a performance.

5.1.1.3 PNG

O PNG é um formato cujas imagens não perdem qualidade, sendo concebido para ter melhores características que o formato GIF. Existem dois tipos de PNG, o PNG-8 e o PNG-24. Cada um dos formatos tem as suas vantagens e desvantagens.

O PNG é a melhor opção para introduzir transparência numa imagem. A sua compressão reconhece padrões horizontais e verticais, sendo o seu algoritmo de compressão mais eficiente que o GIF.

O PNG-8 é a melhor opção para imagens com poucas cores. Suporta um máximo de 256 cores, à semelhança do GIF, e geralmente, tem um tamanho menor.

Os ficheiros PNG-24 não têm restrição do número de cores. As fotografias PNG são 5 a 10 vezes maiores que as imagens JPEG, uma vez que não perdem qualidade [10]. Reduzir o número de cores é benéfico para o tamanho da imagem.

5.1.1.4 Otimização

Com base no estudo dos vários formatos de imagens retiram-se algumas conclusões sobre as situações em que devem ser utilizados os diferentes formatos e as otimizações que devem ser feitas nas imagens com determinado formato, reveladas na Tabela 2.

Formato	Melhor para:	Opções de otimização
JPEG	Fotos e imagens com muitas cores	Redução de qualidade, carregamento progressivo e diminuição de ruído.
GIF	Animações	Reduzir número de cores, aumentar padrões horizontais, reduzir ruído vertical
PNG-8	Imagens com poucas cores	Reduzir número de cores, aumentar padrões horizontais e verticais.
PNG-24	Imagens com transparência parcial	Reduzir ruído e número de cores.

Tabela 2: Formatos de imagem

As imagens, uma vez tratadas, podem sofrer alterações que diminuem o seu tamanho, sem que percam qualidade. O primeiro passo é ajustar a suas dimensões ao tamanho definido na página Web. Uma imagem com dimensões superiores às visualizadas na página Web, tem um impacto negativo no tempo de carregamento dela, pois são transferidos bytes desnecessariamente. As páginas Web são visualizadas em diferentes dispositivos, portanto as imagens têm diferentes dimensões, de ecrã para ecrã. Existe um elemento HTML que permite entregar imagens com diferentes dimensões, de acordo com as características do dispositivo utilizado.

O elemento “picture”, que surge com o HTML5, utiliza as media querys para determinar qual imagem a ser renderizada.

```
<picture>
  <source media="(min-width: 800px)" srcset="big.png">
  <source media="(min-width: 400px)" srcset="small.png">
  <img src="small.png" alt="Description">
</picture>
```

Figura 57: Elemento "picture"

Numa página HTML com o elemento da Figura 57, será renderizada a imagem “big.png” para ecrãs superiores a 800 pixéis. A imagem “small.png” será renderizada em ecrãs superiores a 400 pixéis e inferiores a 800. A imagem apresentada é a primeira cuja condição, descrita no atributo “media”, é cumprida. Os b*rowsers* que não suportam este elemento renderizarão a imagem “small.png”, presente no elemento “img”. O elemento só é suportado pelas versões mais recentes de Firefox, Chrome e Opera. Ainda assim há uma livraria javascript (http://scottjehl.github.io/picturefill) que permite que o elemento tenha o mesmo comportamento em *browsers* que não o suportam.

Após um redimensionamento correto, devem ser utilizados programas que implementam algoritmos de compressão que não afetam a qualidade das imagens. São apresentadas a seguir algumas ferramentas de otimização.

- PNG
  - Pngcrush – o png guarda a imagem em "chunks", alguns deles não são necessários para a visualização na Web, e é seguro removê-los. Esta ferramenta remove os "chunks" não funcionais e minimiza o número de cores disponíveis na imagem.
- JPG
  - Jpegtran – remove comentários, informação para programas de edição de imagem e informação EXIF (data da foto, geolocalização, etc) que são dispensáveis para a visualização da imagem. A utilização desta ferramenta remove qualquer indicação da autoria da imagem, o que é ilegal, por isso deve ser utilizada com cuidado.

Resumindo, a otimização deve ser feita da seguinte forma:

1. Escolher o formato apropriado para a imagem. JPEG para fotografias e imagens com grande variedade de cores e PNG para todo o resto.
2. Reduzir as imagens para as dimensões que são utilizadas na sua renderização, nas páginas.
3. Utilizar ferramentas de compressão que não degradem a qualidade das imagens.

5.1.2 Compressão Gzip

O protocolo HTTP permite comunicar ao servidor se o cliente suporta os métodos de compressão. Quando suportado, os recursos são comprimidos no servidor e o cliente descomprime-os, reduzindo o tamanho dos recursos transferidos. O método de compressão mais popular e eficiente é o Gzip [29]. Outro método conhecido é o *deflate*, mas há um menor número de *browsers* a suportá-lo e os que suportam também descomprimem respostas comprimidas com Gzip.

O cliente Web ao comunicar através do protocolo HTTP/1.1, envia o cabeçalho que indica quais os métodos de compressão que ele aceita.

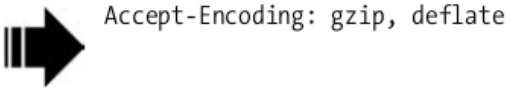


Figura 58: Cabeçalho de aceitação de respostas comprimidas

O servidor ao analisar pedidos com o cabeçalho presente na Figura 58, seleciona um dos métodos de compressão suportados pelo cliente, comprime a resposta e envia-a, anexando a informação do método utilizado. A resposta é descomprimida depois pelo cliente.

Quando o cliente comunica diretamente com o servidor o processo é simples, o cliente recebe o conteúdo comprimido de acordo com os métodos que suporta. A comunicação do

cliente através de um *proxy* complica o processo. O *proxy* é uma máquina que se responsabiliza por fazer os pedidos dos recursos requisitados pelo cliente e devolve-lhe a resposta. Imagine-se o seguinte caso:

1. Um cliente que não suporta Gzip faz um pedido ao proxy.
2. O proxy faz pedido solicitado pelo cliente ao servidor.
3. O proxy recebe resposta do servidor, guarda-a em *cache* e reencaminha-a para o cliente.
4. Existe outro cliente, que suporta Gzip e solicita o mesmo recurso ao mesmo *proxy*.
5. O *proxy* responde com o recurso que foi armazenado em *cache* no primeiro cenário. O conteúdo não teve a oportunidade de ser comprimido, perdendo-se a hipótese de reduzir o tempo de espera por parte do cliente.

Imagine-se que os pedidos são feitos pela ordem inversa, o servidor armazena a resposta comprimida em *cache*, e envia-a para o cliente que não suporta Gzip ou outro método de compressão. O cliente não é capaz de interpretar a resposta, pois não sabe como descomprimi-la. O problema é resolvido no servidor, com o envio do cabeçalho "Vary" juntamente com a resposta. O *proxy* passa a reconhecer se o recurso pode ser recebido ou não pelos clientes que solicitam o pedido. Por omissão, os servidores Apache estão configurados para incluir o cabeçalho "Vary", para evitar o problema mencionado [29].

No servidor configuram-se quais os ficheiros que devem ser comprimidos. A utilização de um método de compressão em ficheiros, cujo conteúdo já foi produzido através de algoritmos de compressão, como as imagens JPEG e os PDF, deve ser evitado, pois a compressão pode aumentar o seu tamanho [29].

A compressão obriga a que os processadores dos servidores e das máquinas clientes gastem recursos a comprimir os ficheiros e a descomprimir. Os ganhos no envio dos ficheiros pela rede, a certo ponto podem não compensar. Os processos de compressão e descompressão podem ser mais morosos que os tempos reduzidos na transferência. Seria necessário considerar várias variáveis como o tamanho dos ficheiros, a largura de banda da ligação e a distância ao cliente, e alguma desta informação não está disponível no servidor. Geralmente, os ficheiros com tamanho superior a 2 Kilobytes devem ser comprimidos com Gzip [29].

Uma experiência realizada por Steve Souders [29], revelou que a utilização de Gzip reduziu 70% do tamanho dos recursos transferidos. Na Figura 59 são revelados os resultados da compressão de vários recursos, utilizando os métodos *deflate* e Gzip.

File type	Uncompressed size	Gzip size	Gzip savings	Deflate size	Deflate savings
Script	3,277 bytes	1076 bytes	67%	1112 bytes	66%
Script	39,713 bytes	14,488 bytes	64%	16,583 bytes	58%
Stylesheet	968 bytes	426 bytes	56%	463 bytes	52%
Stylesheet	14,122 bytes	3,748 bytes	73%	4,665 bytes	67%

Figura 59: Tamanhos das compressões utilizando Gzip e deflate [29]

O primeiro script visualizado na tabela tinha 3.2 Kilobytes e após a compressão Gzip ficou com 1 Kilobyte, menos de 67% do tamanho inicial. A utilização do método *deflate*, originou uma redução de tamanho de 66%. As maiores diferenças apresentadas entre as reduções de tamanho através dos dois métodos utilizados, são no segundo script e segundo

stylesheet (ficheiro de estilos), onde há uma diferença de 6%, sendo maior a percentagem de compressão no Gzip.

A segunda experiência consistiu em analisar os resultados de três cenários. No primeiro sem compressão, no segundo com o documento HTML comprimindo e no último com todos os recursos comprimidos. Revelam-se os resultados na Figura 60.

Example	Components (HTML, CSS, JS)	Total size	Size savings	Response time	Time savings
Nothing Gzipped	48.6K, 59.9K, 68.0K	177.6K	-	1562 ms	-
HTML Gzipped	13.9K, 59.9K, 68.0K	141.9K	34.7K (19.7%)	1411 ms	151 ms (9.7%)
Everything Gzipped	13.9K, 14.4K, 18.0K	46.4K	130.2K (73.8%)	731 ms	831 ms (53.2%)

Figura 60: Resultados dos diferentes níveis de compressão [29]

A compressão de todos os recursos origina um ganho de aproximadamente 831 milissegundos em relação ao cenário onde não há compressão. A configuração do servidor para suportar o Gzip é uma mais-valia para o desempenho da aplicação.

5.1.3 Javascript

A minificação e ofuscação são dois processos que podem ser utilizados para reduzir o tamanho dos ficheiros javascript.

A minificação remove todos os comentários e espaços em branco, como os caracteres de espaço, mudança de linha e tabulação. Os caracteres mencionados são úteis para o programador manter o código organizado no desenvolvimento, mas quando a aplicação está em produção, devem-se remover, para diminuir o seu tamanho.

A ofuscação é uma alternativa otimizada da minificação. Esta remove os comentários, os espaços vazios e diminui o tamanho dos nomes das variáveis e funções. Além de reduzir o tamanho, é ainda utilizado por motivos de segurança.

Segundo Steve Souders [29], a ofuscação tem alguns inconvenientes, uma vez que envolve uma maior complexidade na manipulação de *scripts*, existe um risco maior de serem introduzidos erros. A alteração do nome de funções e variáveis pode não ser conveniente na utilização de APIs, o que obriga a alterar a ofuscação para abrir exceções na alteração de determinados nomes. A minificação é um processo mais simples, pois não intervém na lógica do código, mas limita-se a retirar caracteres que não são necessários para a interpretação do *script.*

Na Figura 61 são apresentados os resultados [29] da compressão entre uma ferramenta de minificação, o JSMin (http://crockford.com/javascript/jsmin), e uma ferramenta de ofuscação, o Dojo Compressor (http://dojotoolkit.org/docs/shrinksafe).

Web site	Original size	JSMin savings	Dojo Compressor savings
http://www.amazon.com/	204K	31K (15%)	48K (24%)
http://www.aol.com/	44K	4K (10%)	4K (10%)
http://www.cnn.com/	98K	19K (20%)	24K (25%)
http://www.myspace.com/	88K	23K (27%)	24K (28%)
http://www.wikipedia.org/	42K	14K (34%)	16K (38%)
http://www.youtube.com/	34K	8K (22%)	10K (29%)
Average	85K	17K (21%)	21K (25%)

Figura 61: Resultados de compressão de um minificador e um ofuscador [29]

As diferenças entre os resultados da utilização das duas ferramentas dependem dos nomes das variáveis e das funções presentes os scripts. Quanto maior for a ocorrência de nomes com elevado número de caracteres, maior será a diferença, uma vez que parte das duas ferramentas envolve as mesmas operações, retirar os espaços brancos. Analisando os resultados, existe uma maior diferença entre resultados no *website* da Amazon, a minificação (JSMin) reduz 15% o tamanho enquanto a ofuscação (Dojo Compressor) reduz 24. Conclui-se que a utilização de um ofuscador para diminuir o tamanho de ficheiros com reduzida dimensão não compensa, devido aos ganhos não serem consideráveis e aumentar o risco de introduzir erros [29].

O Gzip tem um impacto maior na redução do tamanho dos ficheiros, como analisado anteriormente. A utilização das duas técnicas pode reduzir substancialmente o tamanho dos ficheiros javascript. Na tabela da Figura 62 são ilustrados os resultados de três cenários onde é utilizada compressão Gzip em todos. Num é utilizada adicionalmente a minificação e noutro a ofuscação.

Web site	Original size after gzip	JSMin savings after gzip	Dojo Compressor savings after gzip
http://www.amazon.com	48K	7K (16%)	6K (13%)
http://www.aol.com	16K	1K (8%)	1K (8%)
http://www.cnn.com	29K	6K (19%)	6K (20%)
http://www.myspace.com	23K	4K (19%)	4K (19%)
http://www.wikipedia.org	13K	5K (37%)	5K (39%)
http://www.youtube.com	10K	2K (19%)	2K (20%)
Average	23K	4K (20%)	4K (20%)

Figura 62: Impacto do Gzip, minificação e ofuscação

Verifica-se que em média os ficheiros javascript têm os seguintes tamanhos:

- 85 Kilobytes, sem utilização de qualquer método de compressão (Gzip, minificação ou ofuscação).
- 68 Kilobytes utilizando a minificação, reduz 21% do tamanho.
- 23 Kilobytes utilizando a compessão Gzip, reduz 73% do tamanho.
- 19 Kilobytes utilizando a combinação de compressão Gzip e minificação, reduz 78% do tamanho.

A utilização do Gzip tem um impacto maior na diminuição de tamanho dos ficheiros javascript, mas a minificação ou ofuscação, também devem ser tidas em conta. Todos os bytes contam na melhoria do desempenho.

5.2 Reduzir o número de pedidos HTTP

Uma técnica fundamental para reduzir o tempo de carregamento é a redução do número de recursos a serem transferidos. Diminuir o número de pedidos HTTP implica evitar a latência sofrida em cada um deles no estabelecimento da ligação ao servidor e pesquisa DNS.

5.2.1 Combinação de recursos

Um dos tipos de recursos que pode ser combinado são as imagens. Existem várias técnicas para as combinar. O mapa de imagens, por exemplo, pode ser utilizado quando há um conjunto de imagens que se encontram na mesma secção. É possível juntar essas imagens numa só e definir áreas, que ao serem clicadas redirecionam o utilizador para páginas Web diferentes.

```
<img usemap="#map1" border=0 src="/images/imagemap.gif">
<map name="map1">
  <area shape="rect" coords="0,0,31,31" href="home.html" title="Home">
  <area shape="rect" coords="36,0,66,31" href="gifts.html" title="Gifts">
  <area shape="rect" coords="71,0,101,31" href="cart.html" title="Cart">
  <area shape="rect" coords="106,0,136,31" href="settings.html" title="Settings">
  <area shape="rect" coords="141,0,171,31" href="help.html" title="Help">
</map>
```

Figura 63: Mapa de imagens

O código apresentado na Figura 63 diz respeito a uma imagem utilizada num menu de navegação. As *tags* "area", delimitam as zonas cuja interação permite a navegação pelo website. Nelas é indicado o formato, as coordenadas e a página para a qual se navega. Um teste efetuado na utilização deste mapa de imagens, em vez das várias imagens separadas, comprovou um carregamento de página 56% mais rápido (799 milissegundos para 354), utilizando o *browser* Internet Explorer 6.0 com uma ligação DSL de aproximadamente 900 Kilobits por segundo[29]. O senão da utilização desta prática é ser necessário ajustar as coordenadas das várias áreas de hiperligação.

O CSS *sprite* é uma técnica que permite combinar várias imagens numa só, como o mapa de imagens, mas revela-se mais flexível.

Figura 64: CSS sprite de glyphicons

A Figura 64 representa um *sprite*. Na atribuição de estilos, define-se a área da imagem que se pretende que seja renderizada.

```
<div style="background-image: url('a_lot_of_sprites.gif');
            background-position: -260px -90px;
            width: 26px; height: 24px;">
</div>
```

Figura 65: Código de apresentação de uma imagem de um *sprite*

A Figura 65 representa um elemento HTML com uma porção da imagem como fundo. O Bootstrap (http://getbootstrap.com/) e o Fontawesome (http://fortawesome.github.io/) disponibilizam um *sprite* de ícones e um ficheiro CSS com as classes que mapeiam as várias imagens. Os estilos da classe são parecidos com os estilos integrados no HTML da imagem anterior. Quando se pretende utilizar uma porção de imagem escreve-se o elemento HTML da seguinte forma.

A realização da mesma experiência realizada no mapa de imagens, utilizando um *sprite*, apresenta resultados muito semelhantes [29]. Os *sprites* permitem renderizar as imagens em qualquer sítio do documento HTML, enquanto no mapa de imagens, elas são renderizadas todas juntas na mesma secção.

Outra forma mais radical de evitar fazer pedidos HTTP adicionais para obter imagens é incluí-las no HTML, como representado na Figura 66.

```
<IMG ALT="Red Star"
SRC="
lvrKy/FvcPewsO9VVfajo+w6O/zl5estLv/8/AAAAAAAAAAAAAAACH5BAEA
AAsALAAAAAAMAAwAAAQzcElZyryTEHyTUgknHd9xGV+qKsYirKkwDYiKDBia
tt2H1KBLQRFIJAIKywRgmhwAIlEEADs=">
```

Figura 66: Imagem embutida em HTML

A utilização desta técnica tem alguns inconvenientes, tal como o suporte do formato para versões mais recentes do Internet Explorer (http://caniuse.com/#search=data%20uris) e não ser possível armazenar a imagem em *cache*, obrigando a que tenham de ser transferidos os bytes da imagem cada vez que se acede à página. O problema do armazenamento em *cache* pode ser ultrapassado utilizando um ficheiro de estilos com as imagens incluídas, como representado na Figura 67.

```
.home { background-image: url(...);}
.gift { background-image: url(...);}
.cart { background-image: url(...);}
.settings { background-image: url(...);}
.help { background-image: url(...);}
```

Figura 67: Imagens incluídas em CSS

A combinação das imagens utilizadas na experimentação das técnicas de combinação de imagens, apresentam ganhos da ordem dos 50% [29].

Seguir as boas práticas da engenharia do *software* e modularizar o código resulta na criação de vários pequenos ficheiros, o que origina mais pedidos HTTP. Essas boas práticas podem ser utilizadas em ficheiros javascript e CSS. Enquanto se dá o desenvolvimento é do interesse do programador que a aplicação tenha múltiplos ficheiros, por questão de organização de código e de *debug*. Em produção, os ficheiros do mesmo tipo (*script* ou estilos), devem ser combinados num só. A utilização desta técnica num cenário (sem scripts combinados em http://stevesouders.com/hpws/combo-none.php e com scripts combinados em http://stevesourders.com/hpws/combo.php) resultou na melhoria de tempo de resposta de 38%.

As técnicas de combinação revelam-se mais cruciais com o aumento dos recursos a serem transferidos.

5.2.2 *Cache*

Se a redução do número de pedidos HTTP melhora a performance da página, faz sentido colocar conteúdo de ficheiros externos (imagens, javascript e CSS) num documento HTML, sendo preciso fazer apenas um pedido HTTP. A afirmação pode ser verdadeira numa primeira visita à página, depois estar-se-iam a desperdiçar as capacidades que o *browser* tem de armazenar os recursos em cache. O armazenamento de um recurso em *cache* com as políticas adequadas evita que tenha de ser feito um pedido HTTP para o transferir novamente ou verificar se ele foi alterado.

O acesso a uma página HTML com o conteúdo javascript e CSS presente nela, é mais rápido, do que se tivessem de ser transferidos recursos externos javascript e CSS. São comparados dos dois cenários com as características a seguir descritos.

1. Uma a página HTML tem 87 Kilobytes;
2. Uma página com 7 Kilobytes, onde são transferidos ficheiros externos com um total de 80 Kilobytes (um ficheiro de estilos – 59 KB e três ficheiros javascript – 1KB, 11KB e 9KB.

O acesso há página do primeiro cenário é de 30 a 50% mais rápido [29]. Apesar dos resultados, geralmente em casos reais, as páginas que utilizam a abordagem do segundo cenário são mais rápidas, devido ao armazenamento dos recursos em *cache*. As páginas HTML dinâmicas, tipicamente não são configuradas para serem armazenadas em *cache*. A utilização de ficheiros externos, quando configurados para serem armazenados em *cache*, diminuem o tamanho da página a ser transferido e reduz o número de pedidos HTTP.

O estudo do comportamento dos utilizadores face aos resultados de desempenho de acesso às páginas da aplicação ajuda a definir qual é a melhor abordagem a seguir, utilizar ou

não ficheiros externos. Quanto menos visitas de páginas e menor for a recorrência ao *website* por parte dos utilizadores, mais argumentos existem para que não se utilizem ficheiros externos. Um estudo feito pela Yahoo! [29] revela que 75 a 85% das suas visitas, o utilizador tem a *cache* preenchida. Quanto maior for a percentagem de visitas de página com a *cache* preenchida, mais se justifica utilizar ficheiros externos.

Todos os *browsers* suportam o armazenamento em cache [33]. Necessita-se apenas de garantir que o servidor fornece os cabeçalhos corretos no envio das respostas para o *browser* saber como proceder no armazenamento.

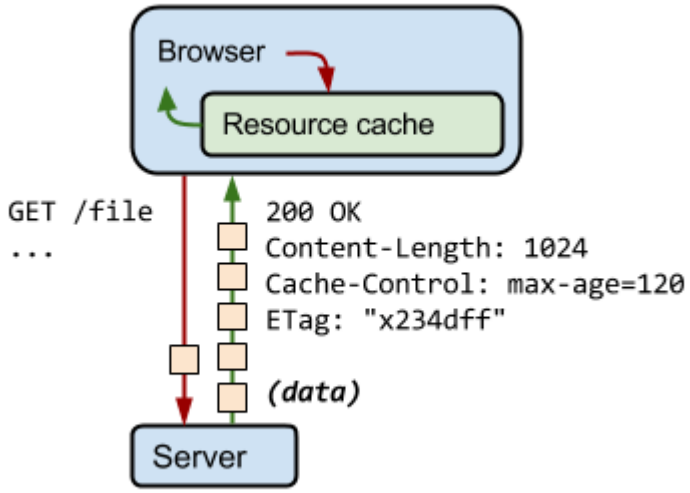


Figura 68: Armazenamento de recurso em *cache* [33]

A Figura 68 ilustra o pedido de um recurso que vai ser armazenado em cache. O *browser* interpreta os cabeçalhos de resposta da seguinte forma:

- **Content-Length -** o recurso tem 1024 bytes;
- **Cache-Control:max-age=120 -** o recurso deve ser guardado em cache durante 120 segundos;
- **ETag -** a versão do recurso permite validá-lo no servidor. A ETag será utilizada posteriormente para verificar se o recurso que está em *cache* é o mais recente.

Ao solicitar o mesmo recurso em menos de 120 segundos, não existe nenhuma comunicação com o servidor e o *browser* utiliza o recurso armazenado em cache. Decorridos os 120 segundos, solicitar o recurso envolve um pedido ao servidor com o cabeçalho "If-None-Match" que contém o valor da ETag do recurso armazenado. A correspondência da ETag no servidor, indica que o recurso não foi modificado. Se não há alterações do recurso não precisa de ser transferido novamente. O servidor responde com o código 304 que indica ao cliente que o recurso não foi modificado, continuando a ser armazenado. Na Figura 69 está o esquema da situação descrita.

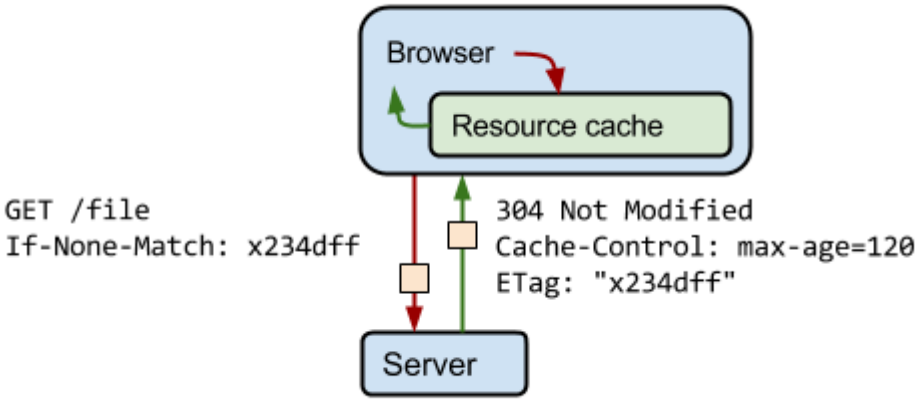


Figura 69: Rearmazenamento de um recurso em *cache* [33]

A ETag, como demonstrado, é um mecanismo que permite aos servidores e *browsers* validar os componentes armazenados em *cache.* Ela é uma cadeia de caracteres única que identifica a versão do componente.

A utilização deste mecanismo contém problemas quando utilizado em determinadas situações. A ETag é gerada de forma única por um servidor específico. Imagine que se utiliza mais do que um servidor para distribuir os componentes da aplicação. O mesmo recurso em servidores diferentes tem ETags diferentes, o que implica que o pedido do mesmo recurso em servidores diferentes origine a sua transferência. Os aspetos tidos em conta na criação da ETag são o tamanho do recurso, a data da última alteração dele e o *inode*. O *inode* é utilizado pelos sistemas de ficheiros para representar o recurso, portanto o seu valor será diferente para cada servidor. A solução consiste em configurar o servidor para que não seja utilizado o *inode* na criação da ETag, nos sistemas Unix. Nos sistemas Windows, os servidores IIS, devem ser configurados para que a diretiva "ChangeNumber" seja a mesma em todos os servidores [29].

Uma configuração correta das políticas pode evitar a latência de vários pedidos HTTP, portanto são descritos a seguir as diferentes diretivas, com as quais se faz o controlo da *cache.* Estas diretivas são associadas ao cabeçalho "Cache-Control", por exemplo, nas figuras anteriores ele tem a diretiva "max-age=120".

- **no-cache** indica que a resposta devolvida não pode ser utilizada sem que haja uma verificação no servidor. Isto é, uma vez que o *browser* tenha armazenado em *cache* o recurso e respetiva ETag, ao ser solicitado novamente, é verificado sempre no servidor se o recurso armazenado foi modificado;
- **no-store** avisa o cliente para não armazenar em *cache* a resposta enviada. Assim é impossível recorrer ao cabeçalho "If-None-Match" e o recurso pedido terá sempre que ser transferido;
- **public** faz com que a resposta seja armazenada, independentemente de existir autenticação HTTP;
- **private** indica que a resposta destina-se apenas a um cliente, não podendo ser armazenada por um intermediário como um *proxy* ou um CDN;

- **max-age** é o tempo de expiração do recurso, especifica o tempo máximo que ele pode ser reutilizado, sem ter de haver verificação de modificações do recurso no servidor.

A adição de qualquer diretiva, na presença de "no-store", não tem qualquer efeito. Assim como utilizar a diretiva "public", quando já está presente o "private". O cabeçalho "Cache-Control" foi introduzido no HTTP/1.1 para ultrapassar dificuldades de outro cabeçalho utilizado até à altura, o "Expires". Este indicava a data até quando o recurso armazenado podia ser utilizado, o que exigia uma sincronização entre o cliente e o servidor. Quando a data de expiração fosse ultrapassada, requeria que houvesse o cálculo da nova data a atribuir. Por outro lado, no "Cache-Control", o controlo é feito com base no número de segundos que o recurso pode ser reutilizado, desde o seu armazenamento. A utilização de mecanismos de *cache* para *browsers* que não suportam o HTTP/1.1, requer a utilização do cabeçalho "Expires".

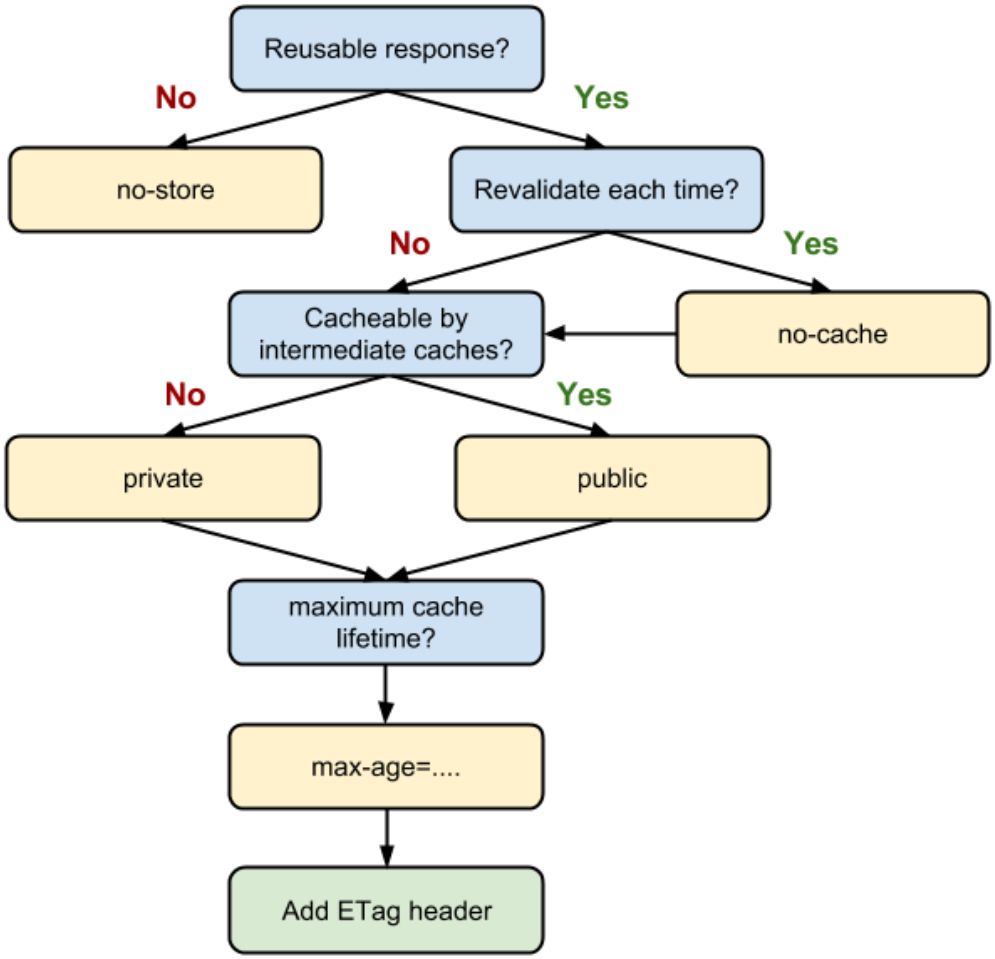


Figura 70: Árvore de decisão das directivas de controlo de *cache*

Ilya Grigorik [33] sugere utilizar a árvore de decisão ilustrada na figura, para configurar a política de armazenamento de *cache*, nas respostas do servidor. O ideal é o cliente armazenar o maior número de respostas pelo máximo período possível e fornecer os *tokens* de revalidação para cada uma, de forma a existir uma revalidação eficiente.

Imagine-se que é feito o controlo de *cache* dos recursos de uma aplicação Web e que se define que um ficheiro de estilos tem um tempo de expiração de um ano. Entretanto esse ficheiro é atualizado no servidor da aplicação pelo programador e existem utilizadores que têm a versão mais antiga do ficheiro. A versão do ficheiro só será atualizada depois do tempo de expiração ou se o utilizador limpar a *cache* do *browser*. Haverá portanto diferenças entre o que é visualizado pelos utilizadores, dependendo da versão do ficheiro que tiverem em *cache.* O problema é resolvido alterando o URL do recurso e corrigindo a página HTML. Por exemplo, no cenário em que são transferidos os recursos da Figura 71.

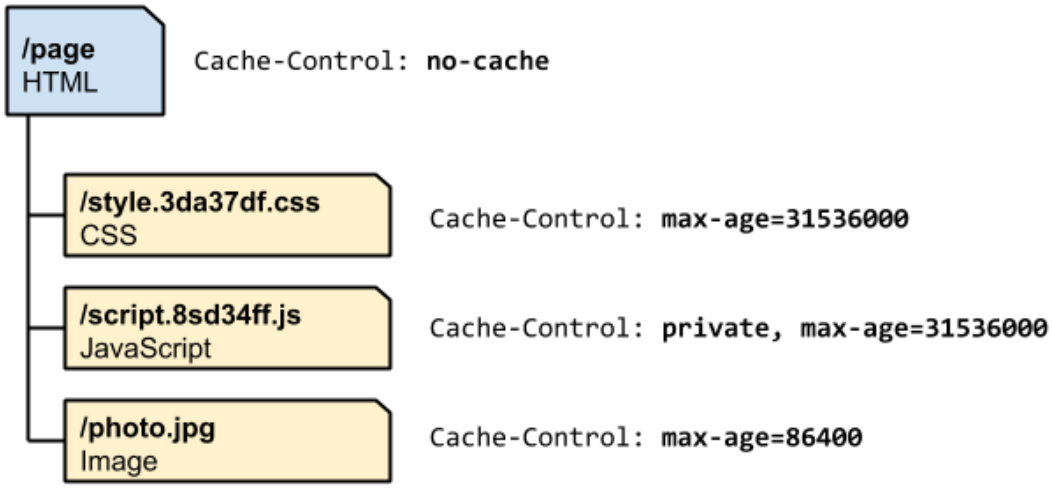


Figura 71: Políticas de controlo de *cache* de recurso de uma página

Cada vez que a página HTML é acedida, é verificado no servidor se ela foi atualizada, uma vez que se utiliza a diretiva "no-cache". Por exemplo, é estabelecido que o ficheiro "style.3da37df.css" deve ser armazenado em cache durante um ano (31536000 segundos). Atualizar o recurso obriga a que o nome dele tenha que ser alterado e depois substituído na página HTML. Assim é gerada uma ETag diferente para a página HTML, esta é obrigatoriamente transferida, e consequentemente o novo ficheiro de estilos é transferido também. A ETag, o controlo de cache e os nomes únicos de URL permitem tempos de expiração longos, o controle sobre onde a resposta pode ser armazenada em cache e a atualização dos ficheiros armazenados.

Steve Souders [29] realizou a experiência e obteve melhorias de performance, ao acrescentar tempos de expiração aos recursos. Os cenários em comparação consistiram na utilização de uma página HTML com seis imagens, três *scripts* e um ficheiro de estilos. Em um ficheiro definiu-se uma política de armazenamento em *cache,* cujos recursos têm tempos de expiração e em outro não, sobre uma ligação DSL de 900 Kbps. No primeiro a página demorou aproximadamente 260 milissegundos a carregar, enquanto no caso em que não há armazenamento em *cache* demorou 600 milissegundos, ou seja, houve uma redução de 57%. A redução é mais notória, quantos mais recursos externos estiverem presentes a página HTML.

Nesta secção abordam-se 3 melhorias de performance. Tempos de expiração longos fazem com que a probabilidade de transferir os recursos seja menor. A utilização de ficheiros externos javascript e CSS facilitam o seu armazenamento em *cache* e, consequentemente, todas as vantagens que isso acarreta para o tempo de acesso à aplicação. Ter vários servidores a tratar

os pedidos de uma aplicação, obriga a que exista uma correta configuração das ETags, de modo a que esta tenha o mesmo valor em todos os servidores, para que não haja transferência do recurso já presente em *cache.*

5.3 Reduzir tempo de latência

A seguir, são apresentadas algumas técnicas de melhoria de desempenho relacionadas com a ligação do cliente ao servidor. É descrito como se pode aproximar fisicamente o conteúdo das aplicações Web dos utilizadores e diminuir o número de pedidos DNS.

5.3.1 CDN

A performance de um *website* é influenciada pela distância física a que os clientes estão do servidor. Quanto mais perto os utilizadores estiverem dos servidores, mais rápido será o acesso à aplicação Web, embora as características da rede possam ter um papel mais crucial na entrega dos recursos. O impacto da distância física aumenta com o número de pedidos HTTP no acesso a uma página Web. Os ganhos com a aproximação aos utilizadores, em cada um dos pedidos, podem ter um grande impacto no tempo de acesso final ao *website*. A aplicação integrada numa rede de fornecimento de conteúdo, mais conhecido por CDN (*Content Delivery Network*), torna possível uma aproximação aos utilizadores.

A CDN é um conjunto de servidores distribuídos por várias regiões geográficas dispersas. Os recursos da aplicação encontram-se replicados em cada um dos servidores. A escolha do servidor que irá fornecer respostas ao utilizador pode ser feita tendo em conta a distância percorrida na rede, que pode ser dada em número de nós onde os pacotes passam, ou com base nos menores tempos de resposta. Além de tempos de resposta mais curtos, existem outros benefícios em utilizar uma CDN. Uma vez que a informação é redundante, há menos risco de perda de informação e é feito o balanceamento de carga de rede quando há algum acesso em massa à aplicação Web.

Na experimentação de um serviço de CDN, por Steve Souders [29], verificou-se que uma página Web carregou 18% mais rápido (1232 milissegundos para 1013 milissegundos). As melhorias são relativas devido à distribuição geográfica dos utilizadores, portanto é difícil indicar quanto mais rápido ficará o acesso. Na loja *online* da Yahoo! [29], registou-se uma redução de 20% na média do tempo de resposta da aplicação aos utilizadores reais.

A utilização deste serviço torna-se mais necessária, quanto mais dispersos estiverem geograficamente os utilizadores do *website*. A internacionalização de um *website*, aumenta a preocupação de procurar soluções para melhorar os tempos de resposta nas várias zonas geográficas.

5.3.2 Redução de pedidos DNS

O papel do DNS (Domain Name System) é mapear os domínios, indicando os seus endereços IP. Os endereços IP são difíceis de memorizar, portanto utiliza-se URLs para aceder

aos *websites*. Além disso, torna-se mais fácil substituir o serviço de uma máquina pelo serviço da outra, bastando para isso atualizar o registo DNS com o IP da máquina substituta. Vários endereços IP podem ser associados ao mesmo domínio, adicionando assim redundância no acesso ao *website*.

A procura DNS tem impacto na performance das aplicações, pois é mais um pedido de rede que tem de ser feito. Associado a ele está o tempo de latência que aumenta o tempo de espera de acesso à página Web. Enquanto o *browser* não tem o endereço IP do servidor, não pode comunicar com ele e encontra-se bloqueado, sem poder transferir nenhum recurso.

O *browser*, assim como o sistema operativo, têm mecanismos para rentabilizar melhor os seus recursos, portanto armazenam em cache os endereços IPs dos domínios acedidos. A maioria dos *browsers* têm a sua cache separada da pertencente ao sistema operativo. A procura do DNS inicia-se com a procura do registo do endereço IP pretendido na *cache* do *browser*. A não existência do IP em *cache* induz a sua procura na *cache* do sistema operativo. Se o IP não existir nas duas *caches*, é solicitado um pedido ao servidor de DNS. Os endereços IP mudam e os seus registos em cache ocupam memória, portanto é conveniente que não fiquem aí guardados infinitamente. A *cache* é periodicamente limpa e são implementadas políticas que vão descartando os registos.

A resposta de um servidor de DNS contém o tempo que o registo pode ser armazenado em cache, designado por TTL (*time-to-live*). O TTL é respeitado pelo sistema operativo, mas os browsers ignoram-no e têm as suas próprias políticas de descarte dos registos de endereços IP. Os *browsers* são configurados para guardar um número limitado de registos durante um período de tempo. A visita, por parte do utilizador, de vários *websites* diferentes num curto espaço de tempo faz com que haja um descarte mais rápido dos registos. O aparecimento da funcionalidade "Keep-Alive", presente no protocolo HTTP, evita que tenham de ser feitos pedidos constantemente ao servidor DNS, uma vez que são reaproveitadas as conexões existentes no *browser*. Salienta-se que os endereços IP ao não estarem na cache do *browser*, podem estar na cache do sistema operativo, por isso ainda é possível aceder a um *website* sem ser um pedido ao DNS.

A técnica de otimização consiste na redução do número de domínios utilizados para transferir os recursos da aplicação, de forma a reduzir o número de pedidos ao DNS. Outra característica do *browser*, que põe de parte a estratégia de utilizar apenas um domínio para a transferência de conteúdo, são os downloads paralelos. A especificação do protocolo HTTP/1.1, sugere que os *browsers* apenas podem fazer download de dois componentes em paralelo por domínio (http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1.4). A utilização de apenas um domínio, bloqueia a transferência de componentes, que ao estarem noutro domínio, podiam ser transferidos ao mesmo tempo, reduzindo o tempo de espera do utilizador. Steve Souders [29] aconselha a utilizar entre dois a quatro domínios.

5.4 Estruturação da página

A transferência de recursos na página HTML é feita à medida que esta é interpretada pelo *browser*. A existência de determinados recursos bloqueia a renderização. A seguir apresentam-

se algumas regras e técnicas de estruturação da página para evitar o comportamento bloqueador na renderização.

5.4.1 Ficheiros de estilos no topo

O CSS é um recurso bloqueador de renderização. No caminho de processamento essencial, uma das primeiras fases é a criação do CSSOM, antes do início da renderização. O *browser* como sistema eficiente que é, não constrói o CSSOM sem que todos os ficheiros de estilos tenham sido descarregados e interpretados. Isto, para que a página não tenha que ser desenhada múltiplas vezes, à medida que os estilos vão sendo interpretados.

Os ficheiros de estilos que são apenas utilizados em dispositivos com determinadas características, não devem ser bloqueadores de renderização em dispositivos que não as têm. Por isso utilizam-se os tipos de "media".

```
<link href="style.css" rel="stylesheet">
<link href="print.css" rel="stylesheet" media="print">
<link href="other.css" rel="stylesheet" media="(min-width: 40em)">
```

Figura 72: Tipos de mídia

Nas *tags* representadas na Figura 72, apenas o primeiro ficheiro de estilos é bloqueador de renderização em todos os cenários. Os outros dois apresentam as condições a serem utilizados na página através do atributo "media". Apenas são bloqueadores quando satisfazerem a condição em "media". Por exemplo, a separação dos ficheiros de estilos para diferentes experiências, como telemóvel e desktop, combinado com a utilização deste atributo pode ter um impacto positivo sobre o desempenho da renderização.

As páginas HTML são interpretadas sequencialmente do topo para o fundo. Os ficheiros de estilos devem ser declarados no topo da página HTML para serem interpretados o mais cedo possível, para que a renderização seja progressiva. As melhorias não se refletem no tempo total de carregamento, mas no início da renderização. A perceção humana é influenciada pelos tempos de espera e é importante dar feedback sobre ações mais demoradas, portanto é essencial que as páginas Web sejam renderizadas o mais rapidamente possível. Jakob Nielsen destaca alguns pontos da importância dos indicadores de progresso [29].

- Assegura ao utilizador que o sistema não deixou de funcionar;
- Indica aproximadamente o tempo que o utilizador deve esperar, incentivando ao utilizador fazer outras tarefas:
- Entretém o utilizador, dando-lhe a sensação que a espera foi menor.

Nas páginas Web, os indicadores de progresso são a renderização dos seus constituintes. É essencial que a página seja renderizada o mais cedo possível para melhorar a experiência de utilização. Colocar os recursos de estilos no fundo da página HTML evita que haja uma renderização progressiva dos elementos, resultando primeiro no aparecimento da página em branco e depois no desenho repentino de todos os elementos da página.

5.4.2 *Scripts* no fundo

O javascript pode alterar todos os aspetos de uma página Web, desde conteúdo, estilos e comportamento dos componentes face às interações dos utilizadores. Por questões de eficiência, os scripts bloqueiam a interpretação do restante conteúdo HTML. A mudança dos elementos DOM e suas propriedades, com recurso a javascript obriga a que haja uma nova renderização de página. A criação do DOM fica interrompida cada vez que é encontrada uma *tag* script. Todos os elementos HTML que estão abaixo de um script, são renderizados após a sua execução, a não ser que para isso sejam utilizados atributos associados às *tags* a serem apresentados mais à frente.

Nos *browsers* mais antigos, que surgiram antes de 2008, o bloqueio efetuado pelos scripts impedia que os recursos a seguir a eles fossem transferidos, evitando os *downloads* paralelos. Assim, garantia-se que os *scripts* eram executados pela ordem correta, um aspeto importante quando existe dependência entre eles. A seguir analisa-se o carregamento de uma página Web que contém dois componentes de estilos representados com a cor verde, três *scripts* a cor-de-laranja e cinco imagens a roxo, como representado na Figura 73.

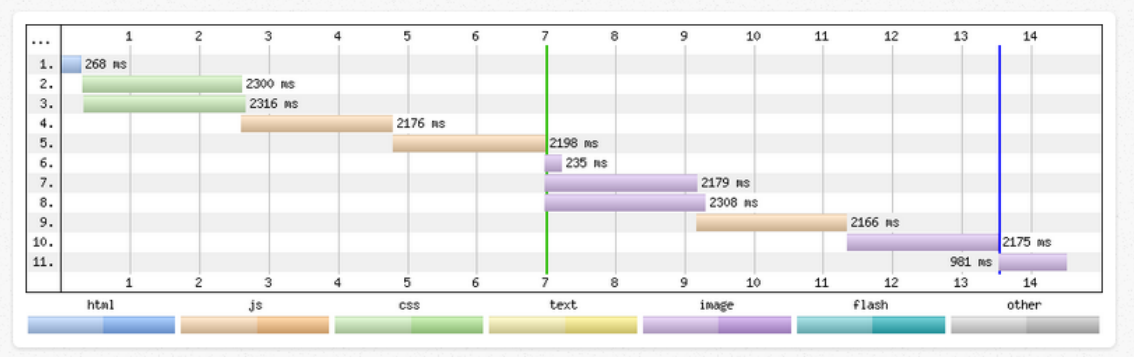


Figura 73: Bloqueio de transferência de recursos por parte de um *script*

Enquanto um script é transferido, há um bloqueio de transferência de recursos que se encontram a seguir. O bloqueio não interfere nos recursos que já começaram a ser transferidos. No fim da transferência e execução do script, são transferidos os restantes recursos.

Seria útil que enquanto o *script* estivesse a ser transferido, os restantes recursos também o fossem paralelamente, sem que houvesse bloqueio. O processo descrito passa a ocorrer com o aparecimento do *pre-loader*. Este passa a ser uma especificação do *browser* em 2008, utilizado pela primeira vez por Internet Explorer, WebKit e Mozilla para aproveitar o tempo de baixa utilização da rede enquanto os *scripts* estavam a ser transferidos e executados [34]. Após a integração no *browser*, a Mozilla reportou que houve melhorias de carregamento de página na ordem dos 19% [35]. A Google ao testar os 2000 sites mais utilizados segundo o ranking Alexa obteve melhorias na ordem dos 20% [36].

O *pre-loader* interpreta o resto da página HTML à procura de outros recursos e transfere-os quando o *browser* inicia a trasnferência de um *script*. Os recursos transferidos a partir de código javascript não tiram partido do *pre-loader*, pois ele apenas procura recursos presentes no documento HTML. A repetição do cenário anterior, num *browser* com o *pre-loader* implementado, resultou no gráfico em cascata representado na Figura 74.

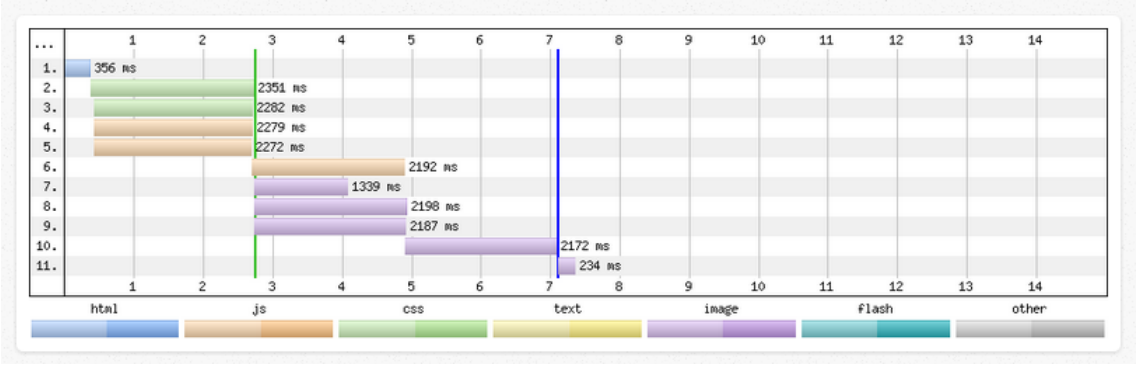


Figura 74: Utilização de *pre-loader* para acesso a uma página Web

Comparando os dois cenários verifica-se que houve uma diminuição de tempo de carregamento, de 14 segundos para 7, ou seja uma melhoria de 50% do tempo de acesso.

O acesso a uma página, com um script no topo da página HTML, atrasa o início da renderização, diminuindo a experiência de utilização, à semelhança do que acontece com os ficheiros de estilos no fundo da página. Por norma, os scripts devem ser posicionados no fundo para evitar o problema mencionado. Nas situações em que isto não é possível, e se pretende evitar o bloqueio, utilizam-se os atributos "async" e "defer" na *tag* script. Eles são suportados na maioria dos *browsers* mais recentes, com exceção do IE8 e IE9, que têm apenas suporte parcial ao atributo "defer" (http://caniuse.com).

```
<script src="//other-domain.com/1.js" defer></script>
<script src="2.js" defer></script>
```

Figura 75: Atributo "defer" num *script*

O atributo "defer", como representado na Figura 75, utiliza-se quando o *script* não altera os elementos HTML ou suas propriedades. Caso seja utilizado e o script altere o documento HTML, não há nenhum problema, mas a página tem de ser renderizada duas vezes, o que aumenta o tempo de carregamento. A sua utilização permite que o resto do conteúdo HTML seja renderizado e o script seja executado no fim da renderização. Todos os scripts declarados desta forma são executados de forma ordenada.

```
<script src="//other-domain.com/1.js" async></script>
<script src="2.js" async></script>
```

Figura 76: Atributo "async" num script

O atributo "async", utilizado nos elementos da Figura 76, surge com o aparecimento do HTML5. À semelhança do "defer", o *browser* assume que ele não vai alterar o conteúdo da página. A renderização continua até que o script seja transferido, e cessa, quando ele é executado. Os *scripts* declarados desta forma não são executados de forma ordenada, o primeiro a ser transferido é o primeiro a ser executado.

Outro comportamento útil seria misturar as duas abordagens. Transferir os *scripts* necessários sem bloquear a renderização e executá-los ordenadamente quando disponíveis. Atualmente não é possível ter esse comportamento apenas com recurso a HTML. O código javascript auxilia a programação desse comportamento.

```
[
  '//other-domain.com/1.js',
  '2.js'
].forEach(function(src) {
  var script = document.createElement('script');
  script.src = src;
  script.async = false;
  document.head.appendChild(script);
});
```

Figura 77: Execução ordenada de *scripts* assíncronos

Os scripts são adicionados a uma fila para serem transferidos e executados. O atributo "async" com o valor *false* evita que a execução seja feita mal os scripts tenham sido transferidos, mantendo a execução ordenada. O script da Figura 77 deve ser adicionado no topo da página. As desvantagens de utilizar este script, são influenciar o tempo de carregamento enquanto está a ser executado e o *pre-loader* não identificar os recursos para transferi-los paralelamente mais cedo. Na decisão de seguir esta abordagem deve-se ter em conta o tempo inicial de renderização e a importância de ter múltiplos scripts a serem executados o mais cedo possível de forma ordenada.

Em suma, os *scripts* devem ser colocados no fundo da página Web. Se não for possível colocar, utiliza-se o atributo "defer" para que não existir bloqueio de renderização e garantir que os *scripts* são executados de forma ordenada. Seguindo a mesma ordem de ideias para não bloquear a renderização, a necessidade de executar um *script* o quanto antes, requer a utilização do atributo "async". No caso de existir a necessidade de executar *scripts* dependentes uns dos outros o mais cedo possível, deve recorrer-se a código javascript para adicioná-los dinamicamente no documento HTML.

5.5 Melhorias na Web 2.0

O termo Web 2.0 surge pela primeira vez em 2004 [29] e pretende marcar a mudança da abordagem de utilização da Internet. Os conceitos desta nova era baseiam-se na utilização de uma interface Web com um agregado de informação que têm como fonte diferentes serviços Web. Trata-se de uma aproximação ao paradigma das aplicações nativas. Os novos conceitos são sustentados por tecnologias como o DHTML e Ajax.

O DHTML permite alterar a página HTML depois de ter sido carregada. Estas modificações são possíveis utilizando javascript para manipular o *Document Object Model* (DOM) no *brower*. Assim, passa a ser possível associar eventos a elementos da página, que quando ocorrem, podem alterar as propriedades dos constituintes da página, como por exemplo, ao passar o rato por cima de uma entrada de um menu, surgir uma lista de subentradas. Outro caso de uso é a substituição completa da página HTML apresentada, sem ter que haver recarregamento completo da página. O Ajax permite obter novo conteúdo sem ter que haver recarregamento da página.

O Ajax evita a habitual quebra de experiência de utilização fornecida pelas aplicações Web tradicionais. Nestas, qualquer ação efetuada obrigava a que fosse feito um pedido ao servidor que resultava no redireccionamento para uma nova página Web. A experiência de utilização é melhorada com a utilização de Ajax porque obtém-se *feedback* sem haver um redireccionamento para uma nova página. A utilização desta tecnologia permite que se obtenha respostas mais rápidas, mas por vezes a sua má utilização pode degradar o desempenho na página.

Algumas das melhorias apresentadas neste capítulo aplicam-se também aos pedidos assíncronos. A utilização de Gzip e minificação reduz o tamanho das respostas a ser transferido. A redução do número de domínios evita que seja perdido tempo na procura DNS e nos estabelecimentos de novas conexões. A redução do número de pedidos, utilizando um serviço que agregue a informação toda da página pode ser uma mais-valia, tendo em conta a capacidade do *browser* de fazer vários pedidos concorrentes. O armazenamento das suas respostas em *cache* pode melhorar consideravelmente o desempenho da página. Nada melhor que utilizar, por exemplo, as ferramentas do *browser* para analisar os tempos dos pedidos assíncronos de forma a identificar problemas de degradação de desempenho.

5.6 Conclusão

As técnicas de melhoria apresentadas neste capítulo são apenas um pequeno conjunto pesquisado. O conjunto escolhido foca-se em técnicas que não requerem a alteração da arquitetura de uma aplicação num estado mais maduro e focam-se principalmente em melhorar os tempos de carregamento no acesso a uma página Web. A pesquisa das técnicas de melhoria passou, sobretudo, pela leitura de duas obras de Steve Souders. Na obra *High Performance Web Sites* [29], são apresentadas 14 técnicas de melhoria de performance, acompanhadas por resultados da sua implementação. Segundo o autor, elas são apresentadas por ordem decrescente de importância, ou seja, pelas que têm melhores resulstados no desempenho.

1. **Reduzir o número de pedidos HTTP,** esta regra diz respeito à combinação de recursos.
2. **Usar um CDN,** também referida neste capítulo.
3. **Adicionar cabeçalho de expiração.** A obra foi escrita em 2007, entretanto já houve alterações no protocolo HTTP, e os novos *browsers* utilizam como controlo dos recursos em *cache,* o cabeçalho "Cache-Control". O suporte da aplicação Web a *browsers* mais antigos, pode levar a que a implementação desta regra ainda se revele importante.
4. **Utilizar Gzip.**
5. **Colocar ficheiros de estilos no topo.**
6. **Colocar *scripts* no fundo.**
7. **Evitar expressões CSS,** a sua não utilização na aplicação a melhorar levou a que a sua exploração não fosse aprofundada.
8. **Utilizar ficheiros externos javascript e CSS,** esta regra é discutida na secção sobre *cache.*
9. **Reduzir o número de pedidos DNS.**
10. **Minificar o javascript.**
11. **Evitar redireccionamentos.**

12. Remover scripts duplicados.
13. Configurar ETags. A técnica é apresentada na secção sobre *cache.*
14. Tornar pedidos Ajax armazenáveis em *cache.*

As obras que contribuíram para a apresentação da secção sobre a melhoria das imagens foram a obra *Even Faster Web Sites* [37], e *Design for Performance* [10] de Lara Hogan.

A performance de uma aplicação não passa apenas pelo rápido carregamento da página Web mas também por analisar o desempenho enquanto se interage com ela. O estudo da performance envolve a análise de tempos de resposta na interação com ela, a memória utilizada, a deteção de uma renderização mais lenta, etc. Tom Baker no seu livro *High Performance Responsive Design* [14], introduz alguns aspetos sobre a análise do desempenho nesse ponto de vista e que ferramentas utilizar para esse efeito. O livro *High Performance Javascript* [38], de Nicholas Zakas, contextualiza sobre o funcionamento da linguagem e sugere um conjunto de boas práticas. O javascript é a linguagem mais utilizada na aplicação desenvolvida que se pretende melhorar. Os resultados da implementação destas boas práticas não foram medidos, mas desde a sua leitura, passaram a ser utilizados para tomar decisões mais acertadas na programação.

No capítulo seguinte é apresentada a aplicação onde se pretendem implementar algumas das melhorias estudadas neste capítulo.

6 Aplicação Teste

Neste capítulo, apresenta-se a aplicação Web onde são implementadas as técnicas de melhoria estudadas. A análise das várias páginas ajuda a identificar situações que precisam de otimização. A plataforma é uma *single page application*, portanto o acesso à primeira página obriga a que sejam transferidos vários recursos da aplicação necessários para a navegação nela.

A plataforma em causa denomina-se Liberopinion e pretende incentivar e facilitar a participação pública no que toca a questões relacionadas com os municípios. As instituições políticas, as autarquias, com a utilização desta ferramenta têm o intuito de apresentar transparentemente as questões relacionadas com as suas políticas, dando a possibilidade aos cidadãos de participarem ativamente nelas. Existem vários processos em que os cidadãos podem intervir. A Liberopinion é uma plataforma em expansão e conta para já com quatro módulos, são eles o "Orçamento Participativo", "Consultas Públicas", "Execução Transparente" e "Conserte Isto". Transversalmente, a plataforma contém as secções de "Notícias", "Agenda" e "Documentos" que informam e auxiliam os eventos dos módulos principais.

A apresentação foca-se apenas no *frontoffice*, embora haja uma maior complexidade no backoffice, onde são geridos os eventos e a informação. É essencial manter as páginas do backoffice com bom desempenho para que os administradores realizem confortavelmente o seu trabalho. Decidiu-se apresentar apenas as páginas do *frontoffice*, pois os utilizadores são a principal razão de existência da plataforma. Os eventos criados pelos administradores não têm valor se não houver participação dos utilizadores, portanto é nas páginas do *frontoffice* que se começa a melhorar a experiência de utilização da plataforma.

6.1 Home

A página de entrada da aplicação, do ponto de vista de desempenho, é a mais importante, pois é por onde a maior parte das sessões se inicia. A espera no primeiro acesso pode levar a que os utilizadores desistam de aceder à aplicação, portanto, as páginas que são utilizadas com mais frequência no primeiro acesso têm uma importância acrescida no que toca à análise do desempenho. A "Home" é uma das páginas utilizadas com maior frequência no primeiro acesso ao *website*.

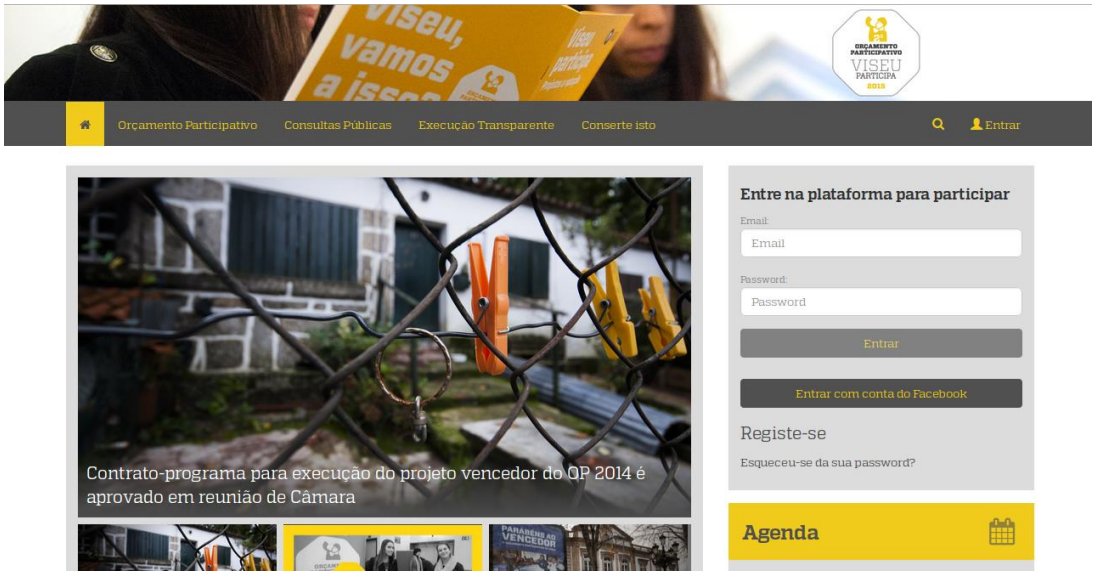


Figura 78: Destaques da página principal

Na Figura 78, do lado esquerdo, a página "Home" contém uma área de comunicação com imagens dos principais destaques relacionados com os eventos da plataforma. Elas servem como hiperligação para outras páginas. Por norma, são destacadas as notícias e os eventos de agenda, mas os destaques podem redirecionar para qualquer página Web, inclusive de *websites* externos.

A coluna direita visualizada na Figura 78, está presente em todas as páginas do *frontoffice*, destacando uma secção para autenticar o utilizador na plataforma. As restantes secções dessa coluna dizem respeito a pontos de ligação às páginas de agenda, de notícias e de documentos. São apresentados também os próximos eventos de agenda, as notícias mais recentes e todos os documentos.

Figura 79: Hiperligações para módulos da plataforma

Na Figura 79 é apresentada uma secção, da página "Home", onde há várias hiperligações para as páginas principais dos módulos. Assim como o menu, é também uma forma de navegação.

As imagens dos destaques são o componente que mais impacto pode ter na página, atendendo à quantidade e dimensões das imagens.

6.2 Orçamento Participativo

O orçamento participativo é um processo de cooperação entre a Câmara Municipal e os cidadãos, inspirado nos valores da democracia participativa. Através dele a população contribui para a tomada de decisão sobre o destino de uma parte, ou de todos os recursos públicos disponíveis, visando a adequação das políticas públicas municipais às necessidades e expectativas das pessoas, para melhorar a qualidade de vida da comunidade.

Cada orçamento participativo pode ter o seu próprio regulamento, descrito pela instituição que o realiza. As instituições podem limitar, por exemplo, a participação à comunidade onde está a ser realizado o orçamento ou permitir que qualquer um participe, passando previamente por um processo de avaliação.

Como referido anteriormente, os orçamentos participativos podem ter diferentes modos de implementação, desde as iniciativas deliberativas até às meramente consultivas. Convém referir que um processo de Orçamento Participativo deve contemplar uma componente digital e uma vertente presencial, de modo a incluir a participação dos indivíduos que, por alguma razão, não têm acesso ou não dominam o uso da tecnologia.

O ciclo de Orçamento Participativo, programado na plataforma, engloba cinco fases. Na página inicial do módulo indica-se a fase atual do orçamento participativo, representada na

Figura 80, algumas propostas ou projetos, dependendo da fase e uma lista das perguntas e respetivas respostas que mais dúvidas suscitam aos utilizadores.

Figura 80: Página inicial do orçamento participativo

A primeira fase consiste na submissão de ideias/propostas por parte da comunidade. As propostas podem ser apresentadas individualmente, através da plataforma *online* preenchendo o formulário da Figura 81 ou em assembleias participativas organizadas pela instituição promotora da iniciativa de Orçamento Participativo (no caso nacional, uma Câmara Municipal ou uma Junta de Freguesia), que fica encarregue de as colocar posteriormente na plataforma, para poderem ser consultadas por toda a comunidade. Note-se, porém, que as propostas são previamente moderadas por uma equipa alocada à instituição proponente, que as publica à luz dos termos de utilização da plataforma ou das normas de participação.

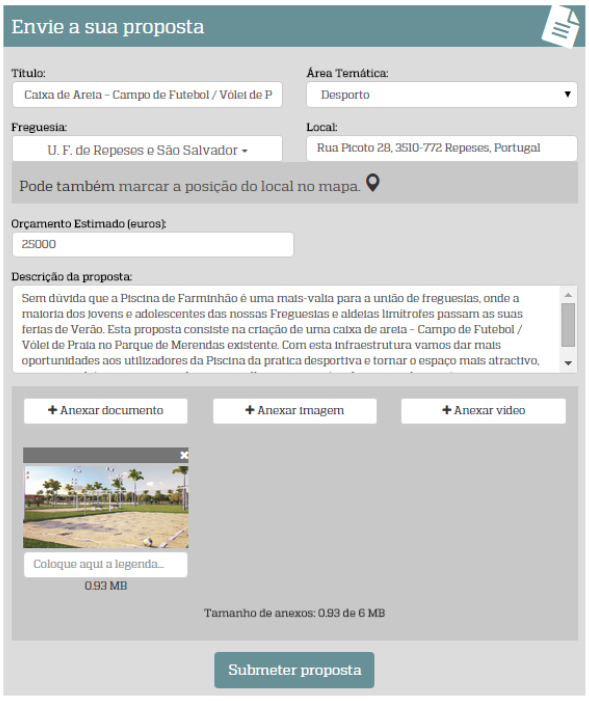


Figura 81: Formulário de submissão de propostas

O utilizador depois de se registar e se autenticar na plataforma, nesta fase, pode submeter as suas propostas, desde que o orçamento participativo tenha sido configurado para aceitar propostas de qualquer utilizador. O orçamento participativo pode ser configurado para aceitar propostas de utilizadores que correspondam a determinados requisitos. O administrador no backoffice, cria um perfil para o sistema, com a informação que pretende pedir ao utilizador e associa-o à ação de submissão de propostas na página do orçamento participativo. Um utilizador que não tenha o perfil, depara-se na página de submissão com o cenário representado na Figura 82.

Figura 82: Hiperligação para pedir perfil

Existe um aviso de que só é possível submeter a proposta depois do utilizador ter o perfil adequado e é fornecida a hiperligação para o pedir. Na página do pedido de perfil são apresentados campos que devem ser preenchidos para obter o perfil em causa.

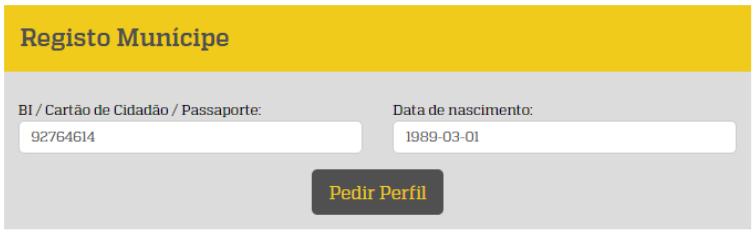


Figura 83: Pedido de perfil

Na Figura 83 é apresentado o formulário de pedido de perfil. Neste caso obter o perfil "Munícipe" requere que seja disponibilizada informação sobre o número de bilhete de identidade, cartão de cidadão ou passaporte e a data de nascimento. O administrador ou outro mecanismo programático, aprovará ou não o perfil segundo a análise da informação disponibilizada pelo utilizador. Após a aprovação, o utilizador pode dar o seu contributo ao orçamento participativo. As propostas são apresentadas numa página de listagem de propostas representada na Figura 84, consoante vão sendo aceites pelos administradores da plataforma.

Figura 84: Listagem de propostas

Na página da listagem de propostas existe um componente com filtros que facilitam a pesquisa de propostas. O clique nos itens da listagem reencaminham para a página com mais detalhes da proposta selecionada. A plataforma integra algumas funcionalidades que facilitam a partilha do seu conteúdo com a rede social Facebook, como ilustrado no canto superior direito da Figura 85.

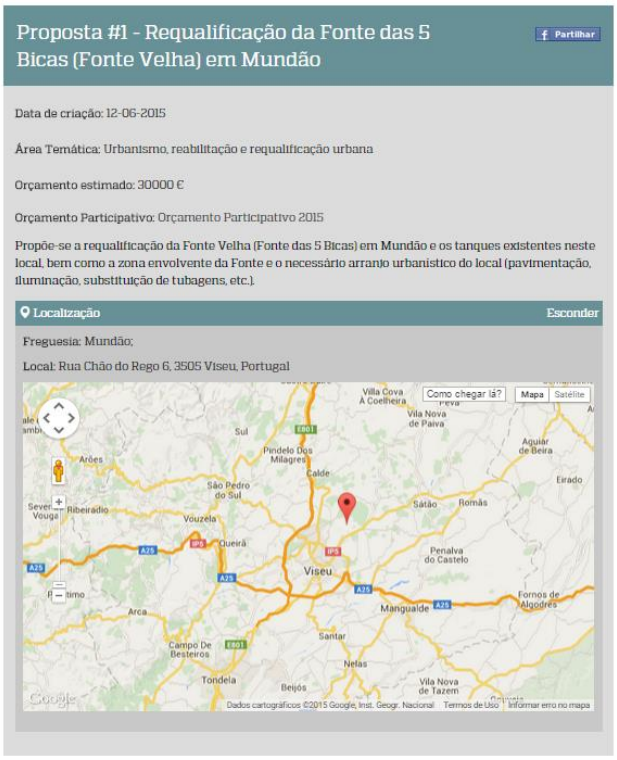


Figura 85: Detalhes de uma proposta

A página dos detalhes da proposta é representada na Figura 85. Ela apresenta a informação submetida pelo seu proponente e opcionalmente pode ter a localização num mapa disponibilizado por uma API da Google. A partir do mapa, se os utilizadores estiverem no município ou arredores e derem autorização para recolher as coordenadas geográficas da sua localização, ao clicar no botão "Como chegar lá?", no canto superior direito do mapa, será

traçado o trajeto para chegarem ao local da proposta. Opcionalmente, podem ser anexados à proposta imagens, documentos pdf e vídeos do Youtube, como apresentado na Figura 86.

Figura 86: Galeria de anexos de uma proposta

Na segunda fase há um segundo processo de avaliação mais rigoroso, tendo como base os critérios de elegibilidade de propostas incluídos e divulgados no regulamento do Orçamento Participativo. Sucintamente, este processo consiste na avaliação de cada proposta como elegível ou não elegível. As propostas que respeitam o regulamento são avaliadas como elegíveis e são transformadas em projetos. Inicialmente o projeto terá os mesmos detalhes da proposta, pois os projetos têm os mesmos atributos que as propostas. Se existirem várias propostas semelhantes ou compatíveis, podem ser agregadas ao mesmo projeto. Por outro lado, caso uma proposta seja avaliada como não elegível, deve ser reportada a razão de não elegibilidade, com base nos critérios de não elegibilidade publicados no regulamento do OP. Nesta fase, a comunidade não pode realizar nenhum tipo de participação, estando apenas disponível para consulta a lista de todas as propostas aceites na fase anterior. Apenas há ações no *backoffice.*

Na fase três é publicada uma lista provisória com os projetos criados a partir das propostas elegíveis e a lista de propostas não elegíveis com as correspondentes fundamentações. No período de tempo em que decorre esta fase, o proponente das propostas avaliadas como não elegíveis pode fazer a sua reclamação, através da página de detalhes da proposta. Ficará visível uma secção com a razão de não elegibilidade, para todos os utilizadores, e um formulário exclusivamente para o proponente, representado na Figura 87.

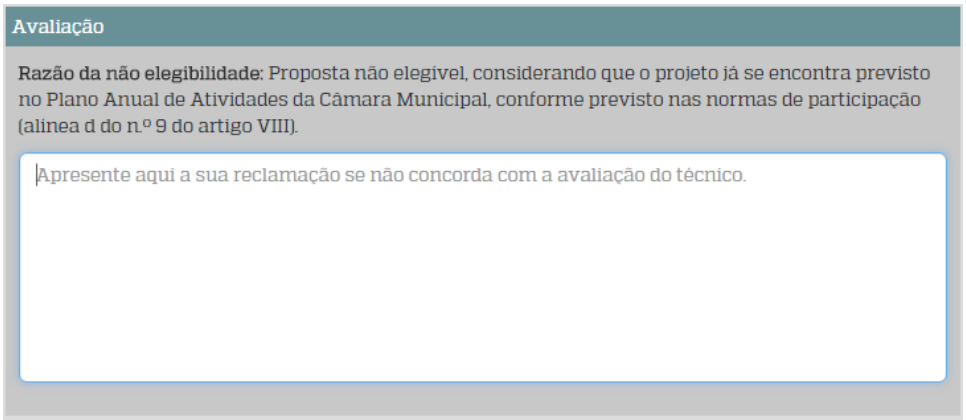


Figura 87: Formulário de reclamação

A reclamação depois de submetida será analisada pela equipa da organização do OP que poderá aceitá-la e transformar a proposta num projeto ou associar a um já existente ou alternativamente, responder à reclamação, mantendo a sua posição. A partir desse momento, a

comunicação entre o emissor da reclamação e a equipa da organização, passa a ser apenas possível, na plataforma, através do envio de mensagens por correio eletrónico.

Existe uma página para visualizar a listagem de projetos. Sendo a sua estrutura muito parecida com a listagem de propostas, como representado na Figura 88, de forma a auxiliar na distinção entre propostas e projeto, os itens da listagem têm uma cor diferente, cor-de-rosa, para os distinguir.

Figura 88: Listagem de projetos

Tanto na página de detalhe das propostas como dos projetos, passa a ser visível uma secção com as propostas ou projetos associados, representado na Figura 89, que servem como hiperligação para a página dos seus detalhes.

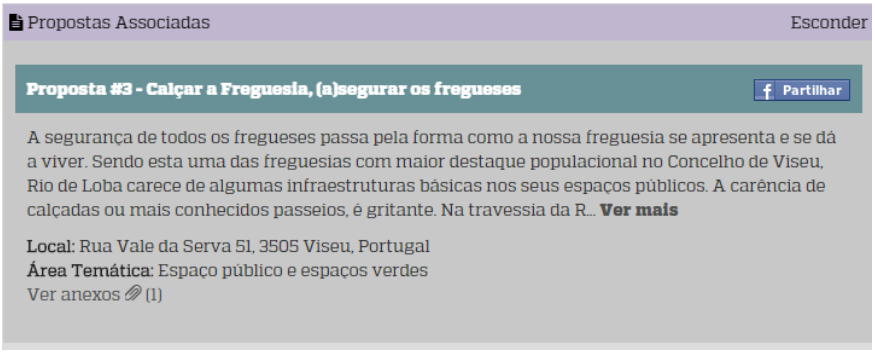


Figura 89: Secção na página do projeto com as propostas associadas

A fase quatro é uma das mais importantes do orçamento participativo. Assim como a fase 1, é dada a oportunidade de participação à comunidade. Nesta fase ocorre a votação nos projetos criados previamente. O projeto ou projetos, de acordo com o regulamento proposto pela entidade organizada, que tiverem mais votos, são declarados vencedores e postos em prática no futuro. A votação pode ser feita, na plataforma, a partir da página de detalhe dos projetos ou na listagem. A restrição base de votação é um voto por projeto. Sem qualquer outra configuração ativa, o utilizador pode votar em qualquer projeto, apenas uma vez. O administrador tem o

poder de configurar o OP, para aceitar um número limite de projetos onde votar ou aplicar os limites dependendo do orçamento estimado dos projetos. Isto é, definir por exemplo, três votos para projetos inferiores a 10,000€ e dois acima do mesmo montante. A listagem de projetos na fase de votação tem o aspeto da Figura 90.

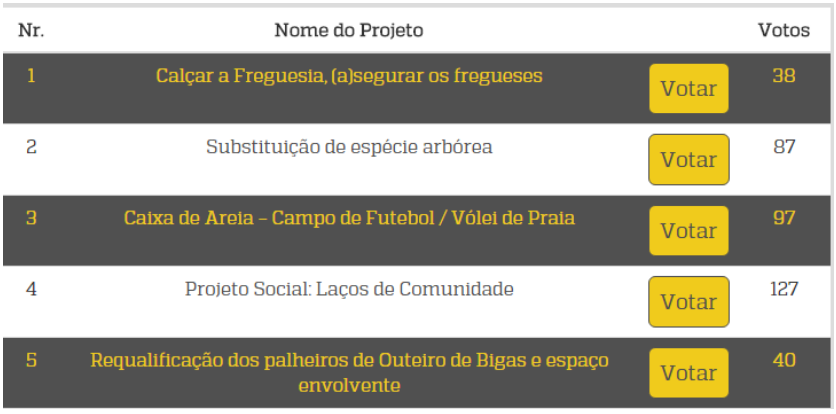


Figura 90: Votação a partir da listagem dos projetos

As entidades responsáveis pela plataforma realizam eventos onde os cidadãos da comunidade comparecem para dar o seu contributo presencial na votação. Os votos são registados no evento, e depois utilizada a conta de um utilizador da plataforma para inserir os votos. O utilizador tem de ter um perfil especial, denominado "Assembleia Participativa", para que não esteja limitado ao número de votos. A Figura 91 apresenta o formulário de submissão para um utilizador "Assembleia Participativa", este introduz num campo de texto o número de votos que deseja submeter.

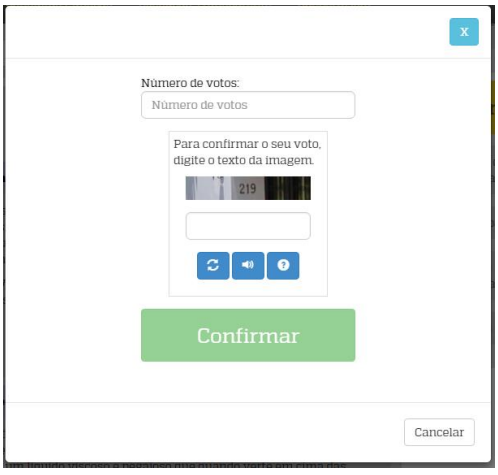


Figura 91: Votação com perfil assembleia participativa

Outro meio é a votação por sms. A plataforma está ligada a um serviço, que recebe SMSs de votação e reencaminha o corpo da mensagem para a plataforma. A partir do corpo da mensagem é extraído o número do projeto e contabilizado o voto. As limitações de votos também se aplicam neste tipo de votação.

A visualização de votos, no *frontoffice*, nesta fase é decidida pelos administradores da plataforma. Permitir ver o número de votos enquanto decorrem as votações têm influência no comportamento dos utilizadores. Se por um lado, a sua visualização influencia os utilizadores

a votar e a incentivar outras pessoas a participar. Por outro pode distorcer a sua decisão na escolha do projeto onde votar. Decidir mostrar os votos no *frontoffice*, informa os utilizadores, dos projetos que vão na frente, como ilustra o ranking de votos da Figura 92.

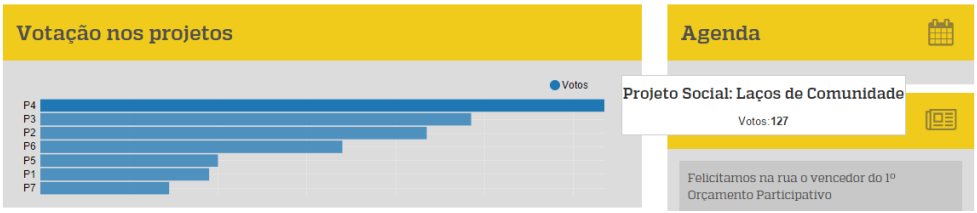


Figura 92: Ranking de votos nos projetos

Disponibilizar informação dos projetos com mais votos pode tornar os utilizadores, interessados na execução de um projeto, altamente competitivos ao ponto de quererem intervir na votação por meios injustos. Os utilizadores com conhecimentos informáticos podem criar scripts que registam utilizadores com contas de *e-mail* falsas e realizem a votação, disparando o número de votos. O mecanismo encontrado para ultrapassar esse problema é a associação de desafios que apenas humanos conseguem resolver, para as ações de registo e votação. Estas são executadas com sucesso quando os desafios são ultrapassados com sucesso. Os desafios utilizados têm o nome de *captcha*. A Figura 93 contém uma imagem com o número 231, que deve preencher o campo de texto para o desafio ser cumprido com sucesso.

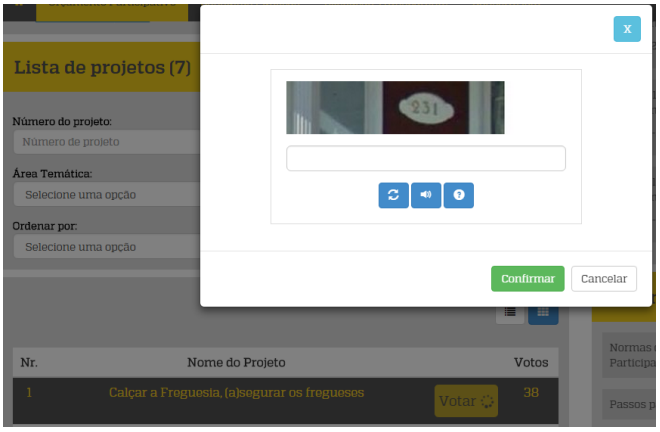


Figura 93: Desafio *captcha* na votação

Ainda assim nada impede que um utilizador crie várias contas, com diferentes emails, e vote no projeto do seu interesse. A solução encontrada é dificultar o processo de votação, restringindo essa ação aos utilizadores com determinado perfil. É visualizado um aviso para os utilizadores pedirem o perfil necessário, como apresentado na Figura 94.

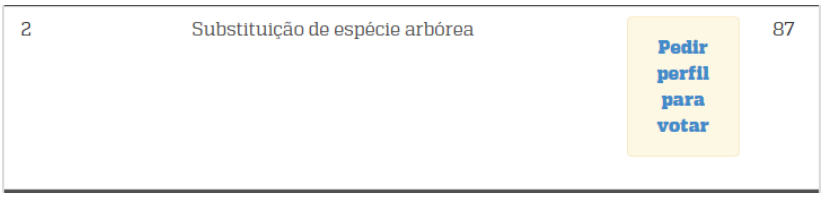


Figura 94: Aviso de necessidade de perfil para votar

No perfil podemos pedir um comprovativo com o número do BI e a partir daí deixar o utilizador votar uma só vez. Assim não pode utilizar várias contas de e-mail para fazer vários votos no mesmo projeto.

O administrador pode bloquear e desbloquear a votação sempre que o pretender através do *backoffice.* Uma vez bloqueadas, pode definir os projetos vencedores. A informação só é visualizada no *frontoffice,* quando o orçamento participativo está na fase cinco.

Na última fase são apresentados os resultados finais da votação, com especial destaque para os projetos vencedores, como apresentado na Figura 95. A definição de projetos vencedores pode ser feita ainda nesta fase. Os administradores devem definir os vencedores de acordo com as regras do orçamento participativo. À partida, os mais votados na fase anterior serão os vencedores.

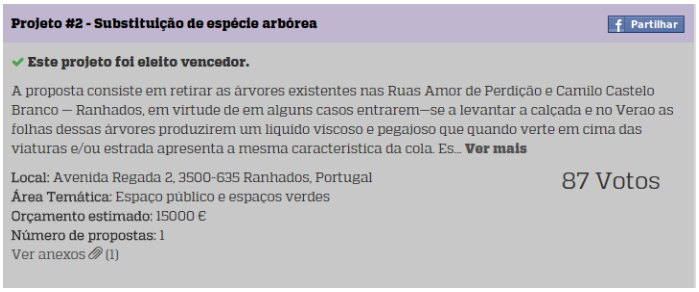


Figura 95: Projeto vencedor na listagem

A autarquia é responsabilizada pela execução dos projetos eleitos como vencedores. O acompanhamento do projeto continua a ser feito na plataforma. Existe uma página que apresenta todos os projetos vencedores dos orçamentos participativos realizados na plataforma. Esta página está presente na escolha da opção execução transparente no menu.

Nos detalhes da página do projeto passa a haver a secção com o título "Acompanhamento da evolução do projeto", onde são acrescentados pontos de situação do projeto. O administrador cria os pontos de situação que são visualizados na página de detalhes do projeto, como representado na Figura 96.

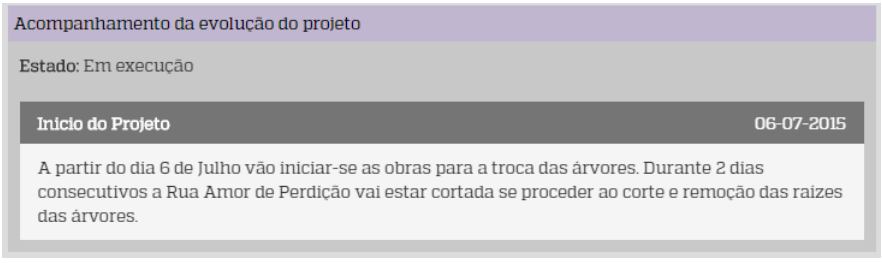


Figura 96: Acompanhamento de evolução do projeto

Ao ponto de situação é possível associar imagens, documentos pdf e vídeos do youtube à semelhança da galeria das propostas e projetos. O utilizador não está interessado em ir todos os dias à página do projeto verificar os seus progressos, portanto há uma funcionalidade na página que permite à pessoa receber por e-mail, mensagens sobre progressos do projeto.

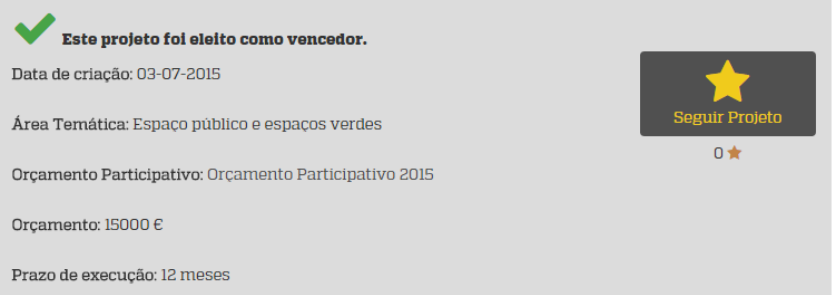


Figura 97: Seguir projeto

No canto superior direito da Figura 97, há um botão, que ao ser clicado por um utilizador, fará com que passemos a seguir o projeto e a receber via e-mail as atualizações do projeto. Debaixo do botão é representado o número de seguidores do projeto.

O Orçamento Participativo termina com o início de outro. As informações do novo Orçamento Participativo passam a estar na página inicial do módulo e as informações do antigo passam a ser acessíveis a partir da página de listagem de orçamentos participativos. A listagem tem o aspeto representado na Figura 98, no mosaico de cada orçamento participativo está o número de propostas submetidas, o número de projetos transformados e a sua designação.

Figura 98: Listagem de orçamentos participativos

Se clicar sobre um item da listagem é redirecionado para uma página onde são visualizadas as listagens de propostas e projetos, com a disposição apresentada na Figura 99.

Figura 99: Listagem de propostas de um OP concluído

6.3 Consultas públicas

A consulta pública é um sistema criado para recolher opiniões da sociedade sobre um tema. Ele permite que a participação pública auxilie o município a formular e definir políticas públicas. A plataforma pretende sustentar este sistema digitalmente, de forma a usufruir da omnipresença e disponibilidade da Web.

Cada consulta pública engloba uma temática sobre a qual se pretende recolher opiniões que podem levar à mudança. Os eventos criados na plataforma podem ser configurados para:

- **Publicar as opiniões** ou não. A sua publicação poderá influenciar positivamente ou negativamente a participação dos cidadãos. Por outro lado, não publicar fará com que o administrador tenha que lidar com opiniões repetidas;
- Permitir o administrador **responder às opiniões** ou não;
- **Moderação das opiniões,** para a sua posterior publicação. Não haver moderação promove a participação, uma vez que os utilizadores têm feedback instantâneo sobre o que fazem no sistema, mas há sempre o risco de serem publicadas opiniões que sejam ofensivas;
- **Votação nas opiniões**, servem para espelhar a aceitação ou rejeição de alguma opinião pelos utilizadores da plataforma. É um indicador para os administradores da plataforma, sobre a questão onde pretendem recolher soluções. Os votos podem ser positivos ou negativos;
- **Percentagem na votação,** à semelhança do que ocorre no Orçamento Participativo, a quantidade de votos de uma opinião pode influenciar ou não a votação. A percentagem dá uma ideia da aceitação ou não de uma opinião, mas não o número de votos ao certo;
- **Mostrar votação** ou não**.**

A página principal das consultas públicas está dividida em quatro secções. No princípio é apresentada uma pequena descrição do módulo. Na Figura 100 são apresentadas as consultas públicas abertas para a participação, depois as últimas consultas arquivadas e uma hiperligação para ver as restantes consultas públicas arquivadas e por fim as últimas participações, submetidas por utilizadores em consultas públicas na plataforma.

Figura 100: Secção com as consultas ativas

Ao selecionar um item da listagem de consultas públicas, há um redireccionamento para a página dos seus detalhes. Se configurada para tal, os utilizadores podem dar a opinião preenchendo o formulário representado na Figura 101.

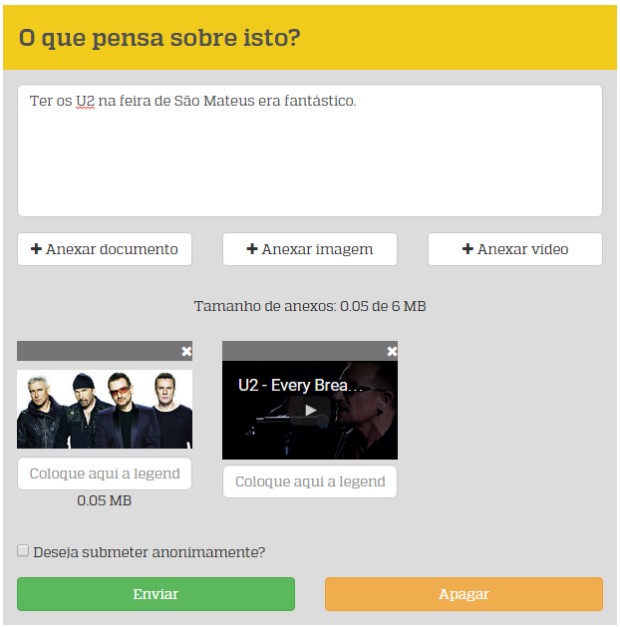


Figura 101: Formulário de submissão de uma opinião

Além da opinião propriamente dita, pode se completada com anexos, que podem ser imagens, documentos e vídeos do youtube. A opinião pode ser submetida anonimamente. Após a moderação do administrador, se assim configurado, a opinião aparece na página dos detalhes da consulta pública, no *frontoffice*.

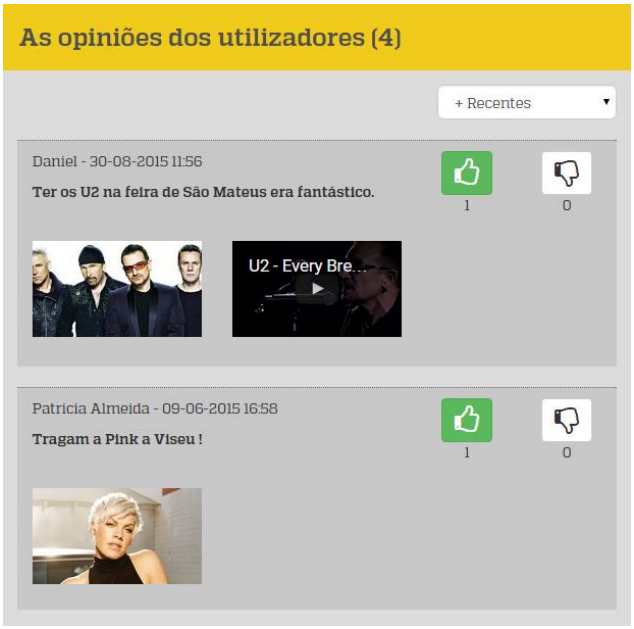


Figura 102: Opiniões de utilizadores

A votação nas opiniões é feita clicando nos botões com polegares, presentes na Figura 102. O campo com o valor "+ Recentes", no canto superior direito da Figura 102, permite ordenar as opiniões. As opções selecionáveis são a ordenação pelas mais recentes, mais votos positivos, mais votos negativos, e melhor diferença de votos, que nós definimos como mais populares.

A participação das consultas públicas, submissão de opiniões e votação, também podem ser restringidas por perfil. Na secção das opiniões aparece um aviso da necessidade de pedir o perfil, como representado na Figura 103.

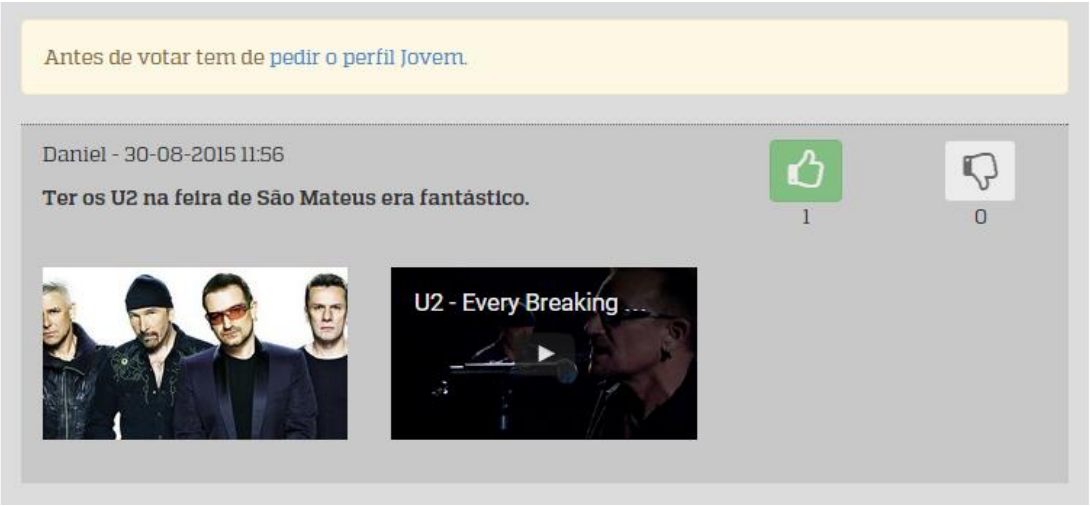


Figura 103: Votação restringida ao perfil

O administrador quando arquiva a consulta pública, esta fica visível na listagem de consultas arquivadas e o seu conteúdo permanece visível na plataforma até que seja eliminada.

6.4 Execução Transparente

Na apresentação do orçamento participativo falou-se de uma página de execução transparente, onde estavam presentes os projetos vencedores. A página de detalhes deles tinha a particularidade de ter uma secção onde eram visualizados os progressos da sua execução. O módulo de execução transparente consiste na partilha de informação sobre projetos que estão a ser executados pela entidade responsável da plataforma. A diferença entre a execução transparente do orçamento participativo e a do módulo, é que numa os projetos derivam de propostas apresentadas pelo cidadão, e noutra eles são criados pelo administrador. É o único módulo onde não há participação dos utilizadores.

As funcionalidades são as mesmas implementadas nos projetos vencedores no orçamento participativo. Os pontos de situação são submetidos no *backoffice* e o utilizador tem a opção de seguir o projeto. A página de apresentação do módulo contém uma secção com os projetos em destaque, um pequeno texto que introduz o módulo, a listagem de projetos acompanhada de um mapa onde estão os marcadores com a sua localização. Clicar sobre um marcador abre uma caixa com uma hiperligação para a página de detalhes do projeto, como apresentado na Figura 104.

Figura 104: Listagem de projetos num mapa

6.5 Conserte isto

O módulo "Conserte isto" tem o objetivo de receber participações da comunidade acerca de situações relativas ao espaço público dos municípios. As situações passam pela reportação de problemas se espaçoes públicos como, por exemplo, degradação de um caminho, jardins mal tratados, veículos abandonados, depositação de lixo em sítio indevido, etc. A existência dum

sistema destes na Web permite que as situações sejam encontradas e analisadas mais rapidamente, não sendo necessárias entidades competentes para reportarem os casos problemáticos.

Na plataforma, os detalhes da ocorrência podem conter a sua localização geográfica e fotografias do problema de forma a identificar mais facilmente os problemas. O formulário apresentado na Figura 105 assemelha-se muito ao formulário de apresentação de propostas.

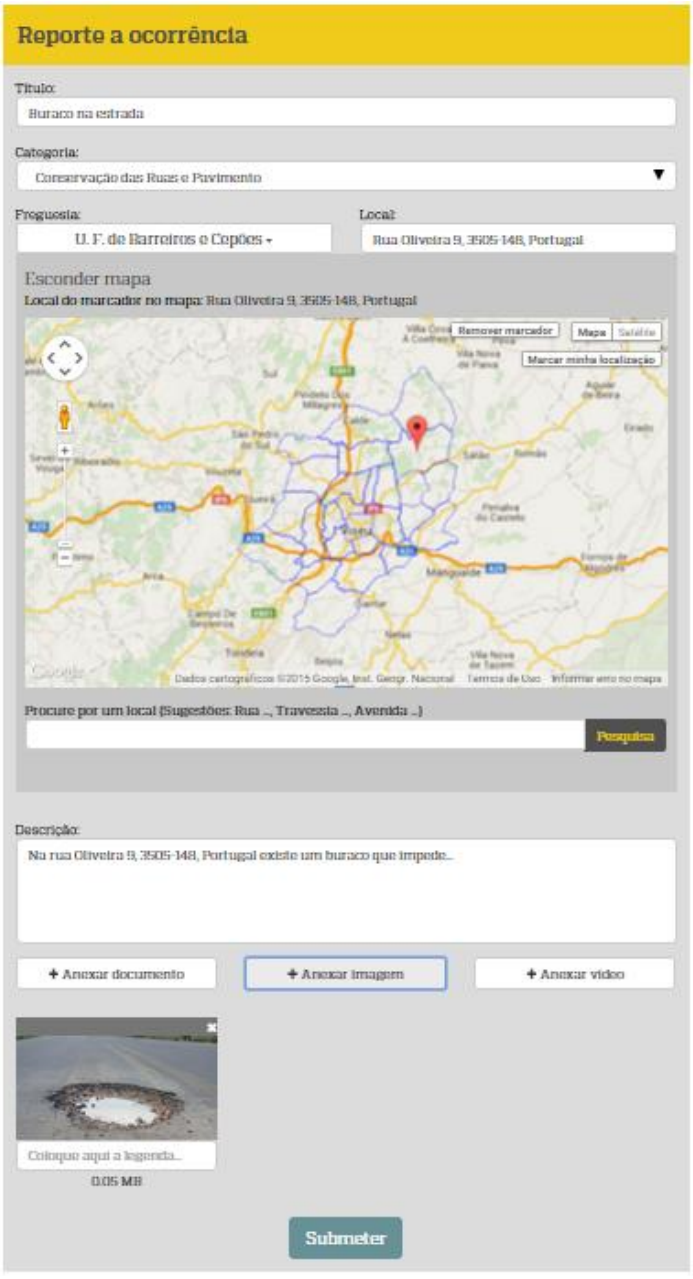


Figura 105: Formulário de reportação de ocorrências

As ocorrências antes de serem apresentadas na plataforma precisam de ser moderadas pelo administrador. Há sempre a possibilidade de publicar ou esconder a ocorrência. Escusado será dizer que interessa ao cidadão visualizar a evolução do solucionamento da ocorrência, e portanto, à medida que vão havendo progressos nesse sentido, são criados pontos de situação

que depois ficam disponíveis no *frontoffice.* O problema quando resolvido, é marcado como tal na plataforma. As ocorrências, à semelhança dos projetos vencedores de um orçamento participativo e dos projetos da execução transparente, também podem ser seguidas. Cada vez que seja criado um ponto de situação, os utilizadores são notificados.

Após a aceitação das ocorrências, elas ficam disponíveis no *frontoffice*, na listagem de ocorrências ou sob a forma de marcadores num mapa, como representado na Figura 106.

Figura 106: Listagem de ocorrências

A pesquisa das ocorrências é facilitada com a disponibilização dos seguintes filtros da Figura 107.

Figura 107: Filtros na procura de ocorrências

6.6 Agenda, Notícias e Documentos

A agenda, as notícias e os documentos servem para auxiliar e complementar a informação disponibilizada nos vários módulos da plataforma. O acesso a elas pode ser feito a partir da secção lateral direita, na maioria das páginas da plataforma.

A área de agenda informa sobre atividades que decorrem no âmbito das iniciativas promovidas nos vários módulos. Na página da agenda, são apresentados os eventos numa visualização em mosaico, estando separados os que já ocorreram, dos que futuramente vão decorrer, como representado na Figura 108.

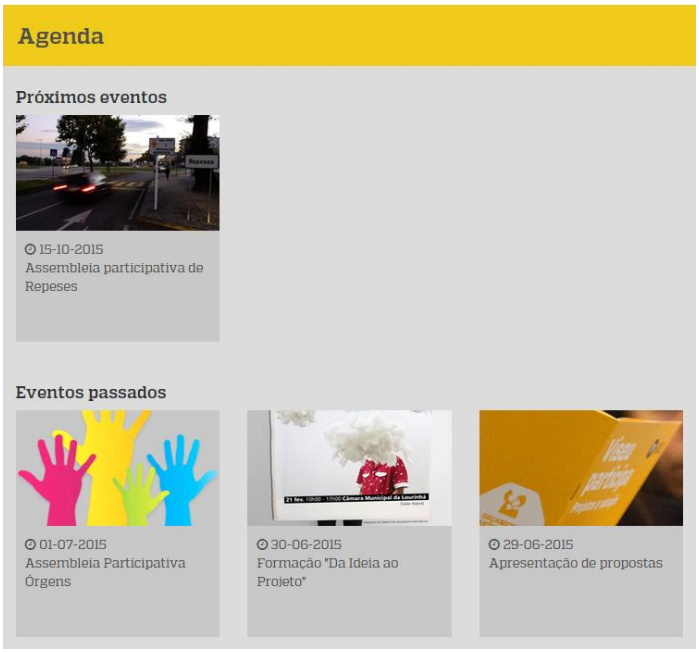


Figura 108: Listagem de eventos de agenda

A degradação do desempenho da página depende da quantidade de informação a ser carregada nela, portanto decidiu-se carregar os eventos por ocorrer e os nove passados mais recentes. Visualizar eventos mais antigos implica que tenhamos de carregar num botão, no fundo da Figura 109, para carregar mais nove eventos.

Figura 109: Carregamento parcial de eventos

Os eventos são formados por um título, texto introdutório, data do acontecimento e opcionalmente por uma imagem.

As notícias divulgam a mais variada informação. A listagem é muito semelhante à listagem de entradas de agenda, como apresentado na Figura 110.

Figura 110: Listagem de notícias

As datas divulgadas dizem respeito à data de publicação das notícias. Também são carregadas nove notícias de cada vez.

Os documentos servem como informação de suporte aos vários processos que decorrem na plataforma. Normalmente são páginas com informação sobre normas ou regulamentos sobre como irá decorrer alguma iniciativa. A sua listagem está em formato mosaico e é visualizado parcialmente o conteúdo de cada documento.

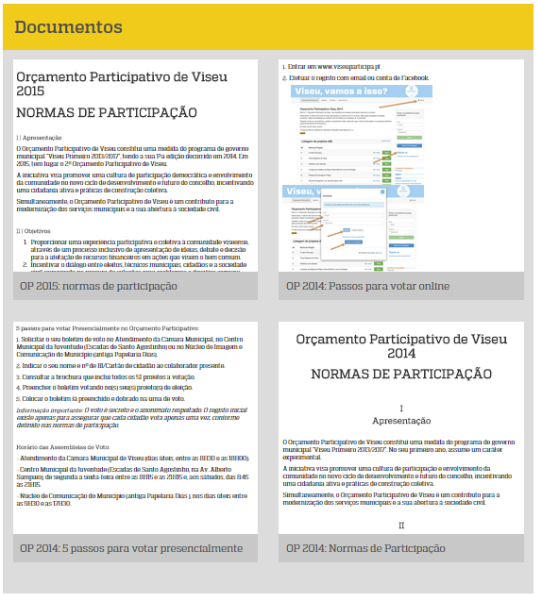


Figura 111: Listagem de documentos

Na página de cada documento há a possibilidade da sua transferência em formato PDF. O clique é feito sobre o texto entre o título e a descrição do documento, como ilustrado na Figura 112

Figura 112: Transferência de documento em PDF

Verifica-se que as páginas das listagens de entradas de agenda e notícias contêm muitas imagens. O elevado tamanho de cada uma delas pode tornar o carregamento da página mais lento. Existe o cuidado de carregar parcialmente as entradas existentes na plataforma, diminuindo o conteúdo a ser transferido e o seu tempo de carregamento.

7 Aplicação de auxílio à performance

Inicialmente pensou-se em desenvolver uma aplicação Web onde pudessem ser feitos testes de performance, implementando técnicas de melhoria de performance pesquisadas. A partir dos dados de desempenho recolhidos na aplicação teste, iriam-se analisar e comparar os resultados entre as várias medidas de otimização. Na análise dos dados recolhidos, escolher-se-iam as técnicas com melhorias mais significativas que seriam implementadas na aplicação web.

No decorrer da pesquisa de aspetos relacionados com a performance na Web verificou-se que a monitorização de resultados de desempenho é uma das etapas mais importantes para a melhoria de uma aplicação Web. Sem quantificar, recorrendo apenas à perceção humana para avaliar o desempenho do *website*, é difícil provar que determinada técnica de melhoria ou conteúdo teve um impacto positivo ou negativo no desempenho da aplicação A medição permite verificar, por exemplo, quais as páginas que têm um tempo de carregamento mais demorado ou qual é o recurso que está a ter mais impacto. Identificar os problemas de performance medindo, torna o processo de otimização mais eficiente. Após o estudo e utilização de plataformas de medição identificaram-se alguns inconvenientes:

- **Versatilidade** - as plataformas focam-se nos testes de monitorização de utilizadores reais ou em testes sintéticos. A certo ponto seria interessante analisar os resultados entre os dois tipos de recolha de dados;
- **Preço** - a maior parte das plataformas dedicadas à medição de performance apresentam planos de serviços pagos. Algumas das plataformas disponibilizam um serviço grátis, mas sem funcionalidades que se poderiam revelar importantes na análise do

desempenho. O WebPagetest é uma solução grátis que não tem limitações no número de testes executados a partir do seu *website*, mas não guarda o histórico dos resultados. O Google Analytics, por outro lado, é uma solução grátis e completa de monitorização de utilizadores reais.

A versatilidade demonstra ser a maior lacuna entre plataformas de medição de performance na Web. A utilização de ferramentas mais completas está dependente do orçamento para investir na monitorização do desempenho. Decidiu-se criar uma plataforma capaz de integrar dados vindos de vários testes sintéticos e de utilizadores reais, e flexibilizar a sua análise e comparação ao longo do tempo. A utilização de APIs e scripts disponibilizados pela comunidade *online* agilizou o processo de desenvolvimento da plataforma.

Ao pesquisar técnicas de melhoria de aplicações Web identificou-se a necessidade de fazer algumas automatizações para auxiliar a otimização de imagens. Portanto, a plataforma está dividida em duas áreas, uma de monitorização e análise de dados sobre o desempenho, onde estão incluídos os testes sintéticos e monitorização de utilizadores reais, e a disponibilização de ferramentas ligadas a aspetos mais técnicos da otimização das páginas Web.

7.1 Análise de requisitos

O principal objetivo da plataforma é avaliar e fornecer ferramentas para melhorar e monitorizar o desempenho das aplicações Web. Anteriormente apresentaram-se os aspetos a ter em conta na avaliação da performance, portanto pretende-se que a aplicação faculte as seguintes ações.

- Avaliar desempenho na rede (*backend*);
- Avaliar desempenho no *frontend;*
- Avaliar o impacto de cada recurso na performance;
- Monitorizar valores das métricas de performance;
- Analisar o desempenho do acesso a um *website*, por parte de utilizadores reais;
- Analisar o impacto das condições dos utilizadores na performance.

A avaliação da rede é feita a partir da análise de três indicadores, o tempo passado na pesquisa DNS, no estabelecimento da conexão TCP e na transferência do primeiro byte. Os três indicadores combinados indicam o tempo de espera passado em processos de rede.

A avaliação de desempenho no *frontend* consiste na análise do tempo de carregamento, o tempo de carregamento total (*Fully Loaded*) e o número e o tamanho dos recursos transferidos.

A avaliação de performance descriminando cada um dos recursos consiste em analisar o tempo de carregamento de cada um deles. A duração deles consiste no tempo de ligação ao servidor e o tempo de transferência. Interessa também referir que deve ser analisado o seu comportamento bloqueador e a transferência dos vários recursos em paralelo. A análise do impacto de cada recurso é feita a partir do gráfico em cascata.

A monitorização dos valores das métricas consiste na análise dos dados recolhidos em testes sintéticos. Estes são executados sempre nas mesmas condições, portanto os resultados são

muito parecidos de teste para teste. As alterações devem-se normalmente a alterações no código da aplicação e do conteúdo dinâmico publicado ou problemas no *website*. Os testes sintéticos são ideias para estabelecer os limites de valores base para as métricas.

Os dados recolhidos dos utilizadores espelham a verdadeira experiência de utilização das páginas Web. As condições de acesso são diversas, o que leva a que os utilizadores tenham diferentes tempos de espera. A avaliação das condições de acesso face aos resultados de desempenho pode identificar problemas e razões de um grupo de utilizadores ter melhores tempos de acesso que outro.

A recolha constante de dados de performance permite que seja visualizada a evolução do desempenho das páginas Web.

7.2 Testes sintéticos

7.2.1 Tecnologias e ferramentas

No desenvolvimento de qualquer funcionalidade o programador depara-se sempre com um dilema, programá-la de raiz ou utilizar ferramentas já existentes para o efeito e opcionalmente alterá-la para cumprir os requisitos. A vantagem de programar a funcionalidade de raiz será a flexibilidade e a melhor adaptação a novos requisitos que possam surgir no futuro, mas isto pode tornar o desenvolvimento mais demorado e o processo de testes mais minucioso. Por outro lado, utilizar uma ferramenta já existente, por norma, a sua integração é mais rápida. Uma extensa comunidade de utilizadores e uma boa equipa de suporte auxiliam a resolução de problemas. As desvantagens são as limitações às funcionalidades fornecidas por elas e a dependência de terceiros para resolução do problema.

7.2.1.1 Phantomjs

Pela flexibilidade e facilidade de utilização, decidiu-se criar um script em Phantomjs, para executar testes sintéticos. O script tinha como objetivo retirar o valor de métricas de performance e detalhes sobre os recursos transferidos como o seu tamanho e duração, e o tempo de navegação total. O objetivo seria criar um relatório similar ao apresentado no Webpagetest, mas sem dependência dessa plataforma.

Para identificar como extrair o tamanho dos vários recursos que são transferidos no acesso a uma página Web considerou-se a utilização do projeto Confess (https://github.com/jamesgpearce/confess). Este projeto não foi utilizado devido à complexidade do formato dos resultados dos testes executados que dificultava a extração da informação sobre o número e tamanho dos recursos por domínio e tipo de recurso.

Alternativamente criou-se um programa, em Phantomjs, que extrai dados de performance que facilmente são processados no servidor e enviados para os utilizadores da plataforma. A condição para detetar o fim de carregamento da página, ao executar o teste com o script, é verificar se não há pedidos de rede durante três segundos. A existência da condição

deve-se ao facto de poderem existir possíveis pedidos de rede com origem na execução de pedidos AJAX, em ficheiros javascript.

Uma particularidade interessante na utilização da ferramenta Phantomjs é a possibilidade de definir o tamanho de janela da execução do teste. Assim é possível simular o teste com dimensões de ecrãs desktop e mobile. O formulário de execução do teste é o apresentado na Figura 113.

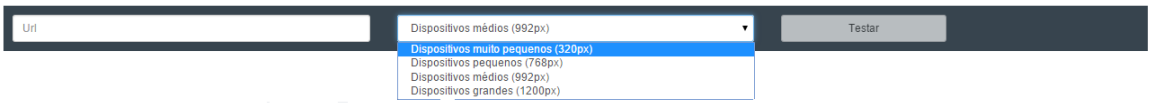


Figura 113: Formulário de execução de teste sintético em Phantomjs

A razão da escolha da divisão das medidas de largura dos ecrãs dos dispositivos (320, 768, 992 e 1200 pixéis), deveu-se ao facto da framework bootstrap (http://getbootstrap.com) utilizar essas dimensões para reajustar os *layouts*. A utilização desta framework permite adaptar mais facilmente o *layout* de um website a dispositivos de dimensões diferentes. Esta está integrada na aplicação web que se pretende otimizar.

Na Figura 114 são apresentados os resultados da execução de um teste com o script programado à página http://google.com.

Figura 114: Resultados dos testes sintéticos

Na Figura 114 estão presentes os valores das três métricas de avaliação de performance, o tempo de carregamento, o número de pedidos de rede e o tamanho total dos recursos transferidos. No teste executado à página http://google.com revelou que o tempo de carregamento da página é de 0.35 segundos e os 11 recursos existentes nela envolveram a transferência de 267.89 KiloBytes.

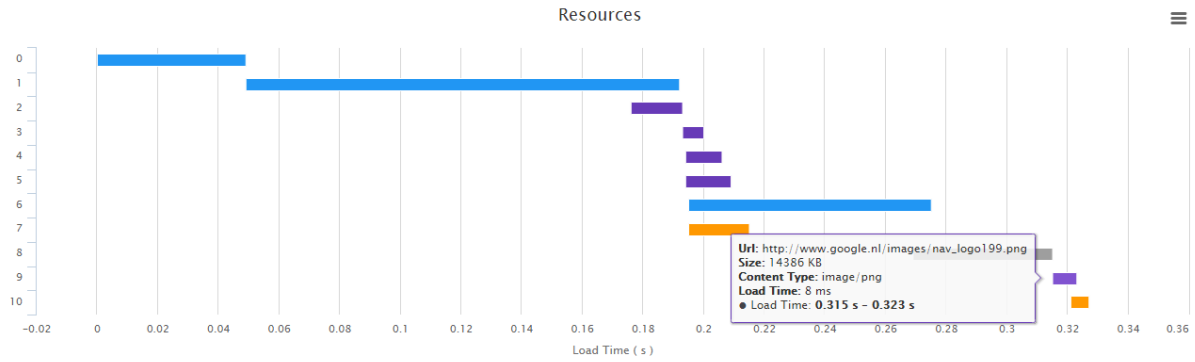


Figura 115: Gráfico em cascata

No gráfico em cascata, da Figura 115, verifica-se a sequência pela qual os recursos são transferidos e o tempo demorado na sua transferência. A cor representada por cada recurso, no gráfico, distingue a sua natureza, por exemplo, o cor-de-laranja indica que são recursos

javascript. Nos detalhes do recurso é apresentado o URL a partir do qual foi transferido, o tamanho, o tipo de recurso, o período de transferência e a duração de *download*. A visualização tabular torna o processo de análise de recursos mais direto, como ilustra a Figura 116.

Id	Url	Content-type	Start Time (ms)	Load Time (ms)	Size (KB)	Status
1	http://google.com/	text/html; charset=UTF-8	0	49	0.26	302
2	http://www.google.nl/?gfe_rd=cr&ei=H-mXVduaL8u3-Qa…	text/html; charset=UTF-8	49	143	59.64	200
3	http://ssl.gstatic.com/gb/images/b_8d5afc09.png	image/png	176	17	9.76	200
4	http://www.google.nl/images/icons/product/chrome-4…	image/png	193	7	1.83	200
5	http://www.google.nl/images/srpr/logo9w.png	image/png	194	12	8.23	200
6	http://www.google.nl/images/srpr/nav_logo80.png	image/png	194	15	35.62	200
7	http://www.google.nl/client_204?&atyp=i&biw=768&bi…	text/html; charset=UTF-8	195	80	0.00	204
8	http://www.google.nl/xjs/_/js/k=xjs.hp.en_US.n4Isy…	text/javascript; charset=UTF-8	195	20	93.35	200
9	http://clients1.google.nl/generate_204		269	46	0.00	204
10	http://www.google.nl/images/nav_logo199.png	image/png	315	8	14.39	200
11	http://ssl.gstatic.com/gb/js/sem_e652ead1622a9a757…	text/javascript	321	6	44.82	200

Figura 116: Vista tabular dos recursos transferidos

Verifica-se, na Figura 116, que o recurso que levou mais tempo a ser transferido foi o que contém o *id* igual a 2 e o que tem maior tamanho é o que possui o *id* 8.

Figura 117: Dados de performance por tipo de recurso e domínio

Existe uma preocupação em distinguir os recursos por domínio e o seu tipo, para identificar a origem (domínio) e natureza do recurso (tipo) que causam problemas de performance. Nas tabelas e gráficos da Figura 117 está descriminada a contribuição de cada domínio e tipo de recurso para o número e o tamanho total de recursos. Do lado esquerdo estão os dados sobre o tipo de conteúdo e verifica-se que o maior número de recursos transferidos são as imagens, apesar de serem descarregados mais ficheiros javascript. Em relação aos

domínios, o maior número e tamanho dos recursos tem origem em http://www.google.nl.

A execução de testes a aplicações no contexto real com este script e a comparação com resultados obtidos através da plataforma Webpagetest levaram à identificação de discrepâncias. O tempo de carregamento do script desenvolvido era muito menor do que nos resultados do WebPagetest como se verifica na Figura 118.

Figura 118: Comparação de resultados em testes sintéticos diferentes

Para investigar as causas desta discrepância decidiu-se fazer um teste na máquina onde é executado o script para verificar a largura de banda da transferência da página. Com isto pretende-se identificar as condições de rede da máquina para encontrar uma explicação para os reduzidos tempos de carregamento.

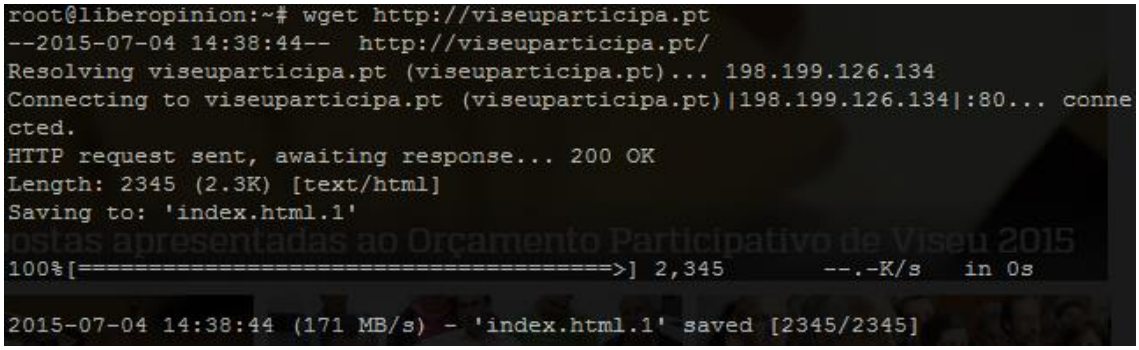


Figura 119: Teste de largura de banda

Ao realizar a transferência de um ficheiro da aplicação com o comando *wget* numa máquina com o sistema operativo Ubuntu comprova-se que o ficheiro foi transferido com largura de banda de 171 MegaBits por segundo, como se apresenta na Figura 119. As condições de rede no servidor são muito superiores às dos utilizadores, portanto os resultados do teste não são os ideais a serem analisados.

De forma a agilizar o processo de recolha de dados dos testes sintéticos decidiu-se integrar a API do Webpagetest na plataforma e adiou-se a resolução do problema.

7.2.1.2 Webpagetest

O Webpagetest disponibiliza uma API REST com a qual se podem executar 200 testes sintéticos diários. A utilização da API alivia o servidor da plataforma em termos de processamento e diminui o número de dados que têm que ser armazenados. A cada teste executado é associada uma chave que identifica o teste. O armazenamento do identificador do

teste permite ver os resultados a partir do *website* do Webpagetest. Uma vez que existe interesse em visualizar a evolução temporal dos valores de algumas métricas, eles foram armazenados na base de dados juntamente com as condições em que foram executados os testes. As condições armazenadas na base dedos são o local a partir da qual é executado o teste, o tipo de tecnologia de rede utilizada, o número de testes e o *browser*. Os valores métricas armazenados foram os seguintes.

- Tempo de procura DNS;
- Tempo de estabelecimento da conexão TCP;
- Tempo para receber o primeiro byte do servidor;
- Tempo de carregamento;
- Tempo de carregamento total;
- Tamanho dos recursos transferidos;
- Número de pedidos de rede;
- Tempo de início de renderização;
- Valor do Speed Index;

Nos valores armazenados distingue-se o cenário em que foram realizados, isto é, na primeira visualização e na visualização repetida onde já existem recursos em *cache*.

7.2.2 Agendamento de testes sintéticos

A plataforma permite agendar a execução de testes. Por questões de facilidade, inicialmente decidiu-se que a execução dos testes agendados seria feita diariamente à meia-noite.

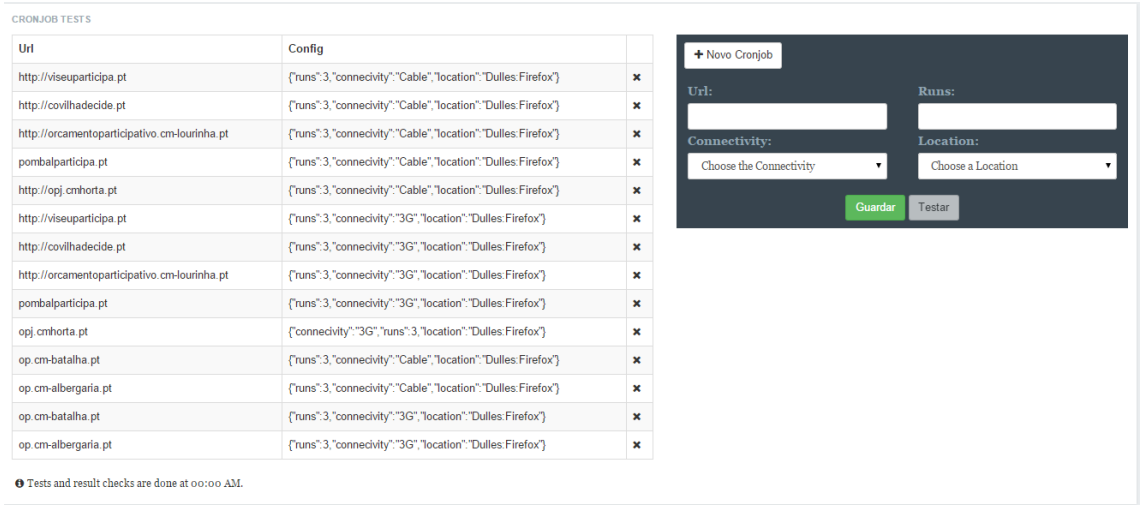
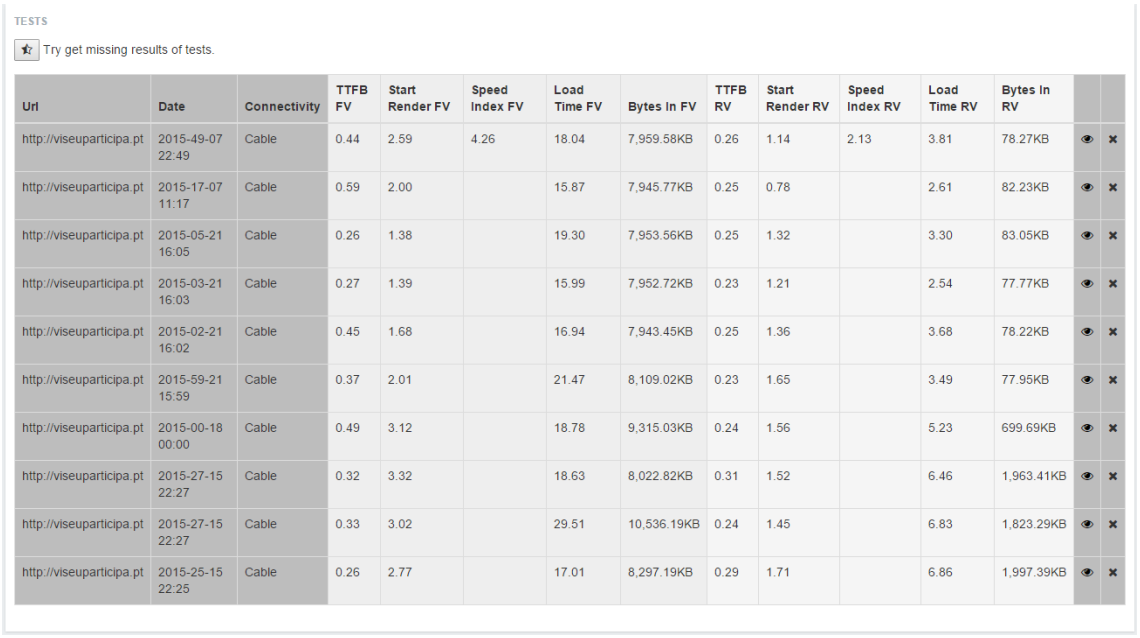


Figura 120: Agendamento de testes sintéticos

Do lado esquerdo da Figura 120 estão os testes agendados e do lado direito o formulário a partir do qual se agenda ou se executa um teste sintético. O formulário é preenchido escrevendo o URL ao qual fazer o teste e selecionando o número de repetições, o tipo de tecnologia de rede e a localização. Na Figura 120 verifica-se um conjunto de testes a vários URLs que são executados 3 vezes de cada vez, com uma ligação de cabo ou 3G, em Dulles, nos Estados Unidos com o *browser* Firefox.

Os testes sintéticos demoram a ser executados. Por esta razão é necessário recorrer mais tarde à API para recolhe-los. A aplicação está programada para recolher os resultados dos testes através da chave de identificação gerada quando é feito o pedido de execução, três horas depois. O tempo que demora a ser executado cada teste é variável e depende da utilização da API por parte de outros utilizadores. Portanto, mesmo depois de três horas, não é garantido que se extraiam os resultados. A verificação só é feita uma vez por dia, portanto a espera pode revelar-se demorosa. Incluiu-se uma opção na plataforma para permitir verificar se os resultados já estão disponíveis em qualquer instante.


TESTS

Try get missing results of tests.

Url	Date	Connectivity	TTFB FV	Start Render FV	Speed Index FV	Load Time FV	Bytes In FV	TTFB RV	Start Render RV	Speed Index RV	Load Time RV	Bytes In RV		
http://viseuparticipa.pt	2015-49-07 22:49	Cable	0.44	2.59	4.26	18.04	7,959.58KB	0.26	1.14	2.13	3.81	78.27KB		
http://viseuparticipa.pt	2015-17-07 11:17	Cable	0.59	2.00		15.87	7,945.77KB	0.25	0.78		2.61	82.23KB		
http://viseuparticipa.pt	2015-05-21 16:05	Cable	0.26	1.38		19.30	7,953.56KB	0.25	1.32		3.30	83.05KB		
http://viseuparticipa.pt	2015-03-21 16:03	Cable	0.27	1.39		15.99	7,952.72KB	0.23	1.21		2.54	77.77KB		
http://viseuparticipa.pt	2015-02-21 16:02	Cable	0.45	1.68		16.94	7,943.45KB	0.25	1.36		3.68	78.22KB		
http://viseuparticipa.pt	2015-59-21 15:59	Cable	0.37	2.01		21.47	8,109.02KB	0.23	1.65		3.49	77.95KB		
http://viseuparticipa.pt	2015-00-18 00:00	Cable	0.49	3.12		18.78	9,315.03KB	0.24	1.56		5.23	699.69KB		
http://viseuparticipa.pt	2015-27-15 22:27	Cable	0.32	3.32		18.63	8,022.82KB	0.31	1.52		6.46	1,963.41KB		
http://viseuparticipa.pt	2015-27-15 22:27	Cable	0.33	3.02		29.51	10,536.19KB	0.24	1.45		6.83	1,823.29KB		
http://viseuparticipa.pt	2015-25-15 22:25	Cable	0.26	2.77		17.01	8,297.19KB	0.29	1.71		6.86	1,997.39KB		

Figura 121: Testes sintéticos Webpagetest

Na Figura 121, está representada uma tabela com os últimos dez testes executados. Verifica-se que no primeiro teste não há resultados em algumas células. A utilização do botão que está no canto superior esquerdo da Figura 121, desencadeia uma verificação da existência dos resultados dos testes com resultados em falta. As células vazias são atualizadas no caso de os resultados estarem disponíveis. Em cada linha da tabela é visualizado o URL, a data de pedido de execução do teste, o tipo de ligação utilizada e valores de algumas métricas. Os valores das métricas dizem respeito à primeira visualização e à visualização repetida. Na tabela são distinguidas as configurações dos testes, os valores das métricas na primeira visualização e os valores das métricas na visualização repetida com tons de cinzentos diferentes. As células com o cinzento mais escuro pertencem à secção das configurações e os valores das métricas na visualização repetida têm o cinzento mais claro. Existem do lado direito uma opção para apagar o teste da base de dados e uma hiperligação para ver os resultados do teste na plataforma Webpagetest.

7.2.3 Evolução do desempenho

O armazenamento dos valores das principais métricas de desempenho em cada um dos testes sintéticos executados, facilita o processamento dos dados no servidor e a visualização da evolução do desempenho de uma aplicação, ao longo do tempo, na plataforma.

Na página dedicada aos testes sintéticos possibilita-se a filtragem dos dados por intervalo de tempo, página Web, tipo de tecnologia de rede e o cenário (primeira visualização e visualização repetida) através do formulário representado na Figura 122.

Figura 122: Formulário de seleção de dados dos testes sintéticos

Os dados apresentados na plataforma estão armazenados na base de dados, existindo assim uma dependência menor da API Webpagetest, que por razões desconhecidas pode não estar acessível. Uma vez selecionado o URL do formulário da Figura 122, são apresentados os resultados dos testes.

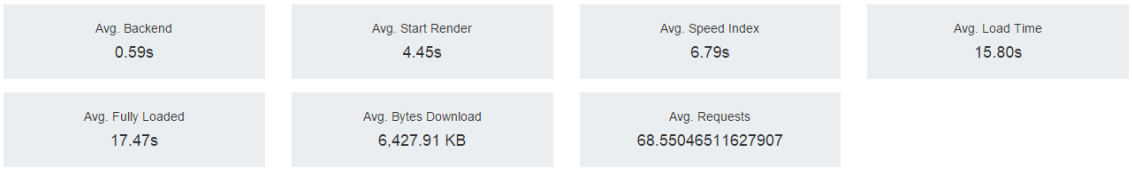


Figura 123: Média Global do valor das métricas

Na Figura 124 são ilustrados os valores médios das principais métricas de desempenho (*backend*, início da renderização, Speed Index, tempo de carregamento, tempo de carregamento total, tamanho dos recursos transferidos e o número de pedidos de rede).

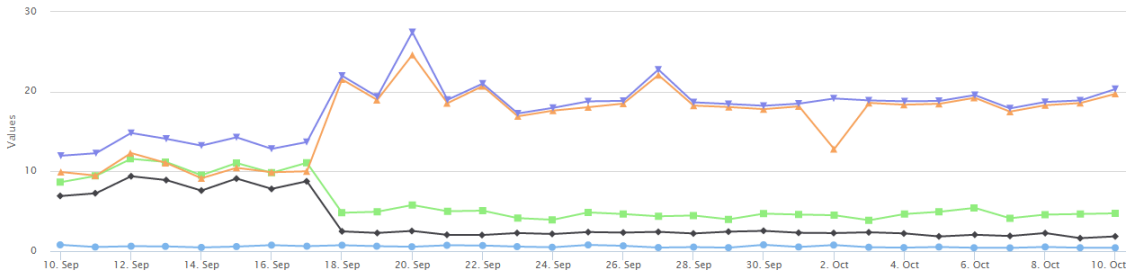


Figura 124: Resultados dos testes sintéticos do Webpagetest

No gráfico da Figura 124 está representada a evolução temporal dos valores das métricas que têm como unidade de medição o tempo. É visualizada a evolução do tempo de *backend*, início de renderização, Speed Index, carregamento e carregamento total.

7.2.4 Comparação com outros *websites*

A comparação dos valores das métricas entre *websites* concorrentes é útil para estabelecer a prioridade de otimização de desempenho. Se os utilizadores deixam de utilizar um

website porque os seus tempos de resposta são demorados e passam a fazer mesmas tarefas num *website* concorrente que tem melhores valores de desempenho, existe uma urgência em otimizar o desempenho.

Na aplicação teste o caso de utilização é diferente. A plataforma "Liberopinion" está preparada para ser utilizada em vários *websites*. O desenvolvimento contínuo da plataforma obriga a que exista uma atualização de todos os *websites* que a utilizam de forma a corrigir erros e a acrescentar funcionalidades. Uma vez que o código da plataforma é igual em todos os *websites* e o seu alojamento é feito no mesmo fornecedor do serviço, espera-se que tenham desempenhos similares. A diferença entre os *websites* é o conteúdo que os administradores e os utilizadores submetem. A comparação entre eles auxilia a identificação de problemas de desempenho, estabelecendo prioritariamente os *websites* a serem otimizados.

Na Figura 125 é representado um gráfico de evolução temporal dos valores de uma métrica selecionada em diferentes *websites* analisados.

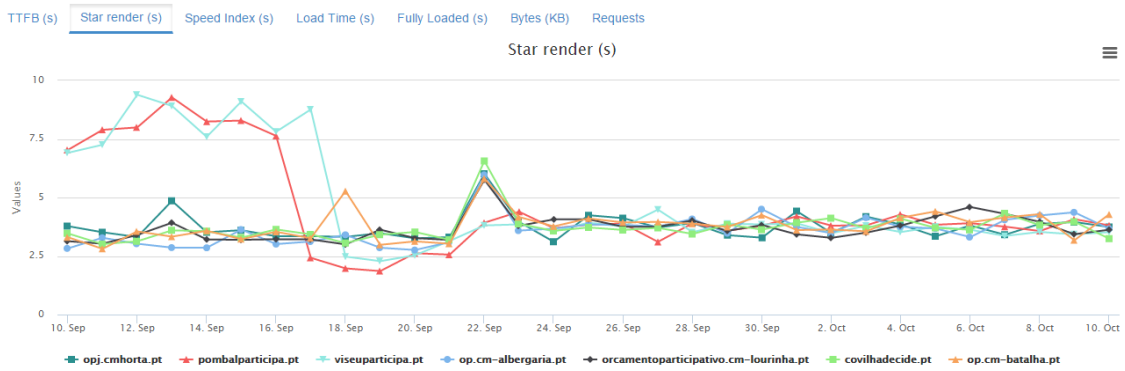


Figura 125: *Benchmark* de testes sintéticos

No gráfico estão representados os valores de início de renderização em cada um dos dias que foram executados testes sintéticos às páginas iniciais de sete *websites*. As métricas selecionáveis são o tempo para receber o primeiro byte, o tempo de início da renderização, o Speed Index, o tempo de carregamento, o tempo de carregamento total, o tamanho dos recursos transferidos e o número de pedidos de rede.

7.3 Monitorização de utilizadores reais

A monitorização do desempenho em utilizadores reais requer modificações no código dos *websites* onde se pretende recolher informação da performance, ao contrário do que acontece nos testes sintéticos. A monitorização é feita passivamente, só existirão dados de performance se houverem visitas de utilizadores reais.

7.3.1 Recolha dos dados

O processo de recolha de dados utilizado consiste na transferência de um ficheiro javascript por parte de um utilizador, com fonte na plataforma que se apresenta. Na execução desse ficheiro, recolhem-se e processam-se os dados de desempenho, que posteriormente são enviados para o servidor da plataforma, onde são armazenados na base de dados.

O ficheiro que analisa e processa os dados sobre o desempenho tem como base o projeto *open source* boomerang (https://github.com/yahoo/boomerang), desenvolvido por programadores da Yahoo. Utiliza-se um *plugin* que recorre à *Navigation Timing API* para recolher dados de uma forma mais precisa. A *Navigation Timing API* deteta os momentos em que ocorrem os vários eventos do processo de carregamento de página. Na análise de desempenho consiste em averiguar a duração dos processos envolvidos no carregamento da página. De forma a facilitar o processamento dos dados por parte do servidor, foram feitas alterações no script para que fossem enviados o número de recursos transferidos e os dados da duração dos seguintes eventos:

- Procura DNS;
- Estabelecimento da ligação TCP;
- Recebimento do primeiro byte, depois de estabelecida a ligação TCP;
- Descarregamento do resto do conteúdo inerente à página acedida;
- Latência, o tempo passado no *backend*, que envolve os processos desde a procura DNS até ao descarregamento de todos os recursos associados;
- Carregamento da página no *frontend*, que inclui os processos de renderização da página e execução de ficheiros javascript;
- Processo completo de navegação.

No acesso à página de uma aplicação Web monitorizada, a transferência do script que faz a medição degrada o desempenho. Existe necessidade de utilizar técnicas que suavizem o seu impacto da transferência do script de medição dos tempos de carregamento dos utilizadores reais. Face ao estudo de técnicas de melhoria [37], concluiu-se que o script devia ser carregado assincronamente para não bloquear a transferência de outros recursos. O script encontra-se num domínio diferente da aplicação avaliada e não há interesse em preservar a ordem pela qual é transferido, portanto recorreu-se a uma técnica que utiliza javascript para adicionar a *tag* no documento que desencadeia a sua transferência. O código representado na Figura 126, transfere o script de monitorização colocando a sua referência no topo da página HTML.

```
<script>
  (function(d, s) {
    var js = d.createElement(s),
    s = d.getElementsByTagName(s)[0];
    js.async=1;
    js.src="http://performance.liberopinion.com/api/rum/analytics.js";
    s.parentNode.insertBefore(js, s);
  }(document, "script"));
</script>
```

Figura 126: Pedido de script de análise de performance

A técnica é utilizada também na transferência do script do Google Analytics para recolher dados sobre as visitas dos utilizadores. Comprova-se no script representado na Figura 127, que há utilização javascript para incluir a *tag* que desencadeia a transferência do script e o envio de dados sobre as visitas para os servidores do Google.

```
(function(i, s, o, g, r, a, m) {
    i['GoogleAnalyticsObject'] = r;
    i[r] = i[r] || function() {
        (i[r].q = i[r].q || []).push(arguments)
    }, i[r].l = 1 * new Date();
    a = s.createElement(o),
        m = s.getElementsByTagName(o)[0];
    a.async = 1;
    a.src = g;
    m.parentNode.insertBefore(a, m)
})(window, document, 'script', '//www.google-analytics.com/analytics.js', 'ga');
```

Figura 127: Pedido de script de análise de visitas do Google Analytics

O pedido de rede HTTP enviado com a informação relativa ao desempenho da aplicação, transmite também o endereço de rede (IP) do utilizador e as características de acesso através do "User-agent".

O projeto freegeoip (https://freegeoip.net) disponibiliza uma API REST para identificar a localização do utilizador através do seu endereço de rede, o IP. A API está limitada a 10000 pedidos de rede por hora mas, à semelhança do que acontece para o Webpagetest, o problema de limitação pode ser ultrapassado instalando e executando o projeto num servidor privado.

A informação denominada "User-agent" identifica o sistema operativo e o *browser* utilizado na visita. Através do sistema operativo pode supor-se que tipo de dispositivo está a ser utilizado, por exemplo, se o sistema operativo for Android, iOS ou Windows Phone pressupõe-se que o dispositivo utilizado é um smartphone ou tablet.

Torna-se possível assim categorizar os dados, por exemplo, por região geográfica, *browser* ou sistema operativo.

7.3.2 Visualização dos dados

A plataforma tem uma página dedicada à análise dos dados recolhidos de utilizadores reais. Os dados recolhidos permitem ter uma noção do desempenho que os utilizadores têm ao aceder às páginas dos *websites* analisados. Na monitorização em causa não há distinção entre os acessos em que existem recursos ou endereços IP em *cache* e há reutilização de ligações TCP.

A avaliação do desempenho das páginas Web, no acesso dos utilizadores, inicia-se selecionando a amostragem de dados que se pretende analisar, escolhendo o período de tempo e o *website* através do formulário apresentado na Figura 128.

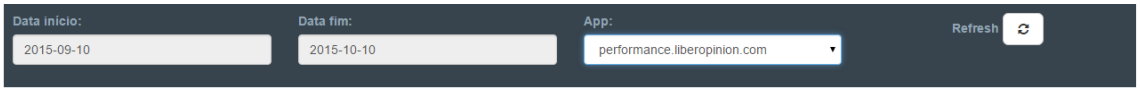


Figura 128: Formulário de seleção de dados de desempenho de utilizadores reais

À semelhança da página de análise de dados dos testes sintéticos são apresentados os valores médios das métricas de performance, como apresentado na Figura 129.

Figura 129: Valores médios do desempenho em utilizadores reais

A própria recolha de dados de desempenho tem impacto no desempenho, portanto foram recolhidos apenas os valores globais de carregamento da página, sem analisar o desempenho de cada recurso transferido. A informação representada na Figura 129 informa que das 5257 amostras recolhidas, o tempo médio despendido no *backend* (procura DNS, ligação TCP e tempo para receber o primeiro byte) foi de 0.46 segundos, no *frontend* foi de 10.84 e o tempo médio de carregamento foi de 11.64 segundos. Verifica-se que cerca de 93% do tempo de carregamento é passado no *frontend*, comprovando uma vez mais a regra de ouro da performance mencionada por Steve Souders [32].

A evolução diária do valor das três métricas e do número de amostragens é representada num gráfico como o da Figura 130.

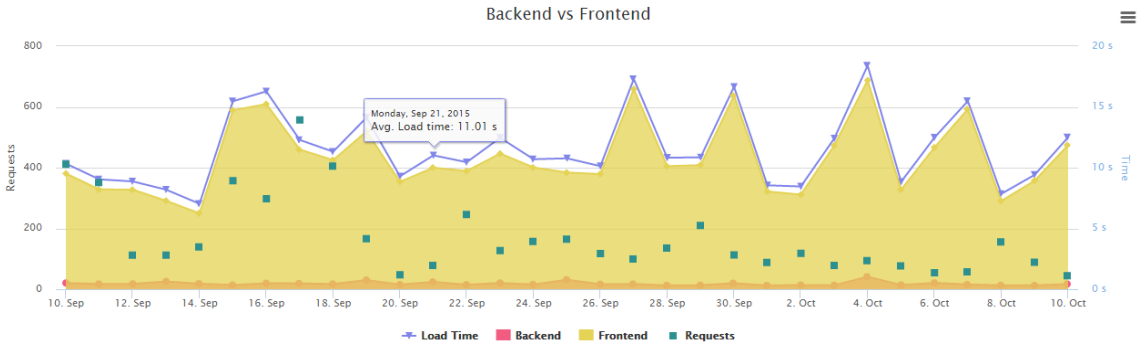


Figura 130: Impacto do *frontend* e *backend* em utilizadores reais

O número de amostras está em forma de pontos e a escala dos seus valores está representada no eixo vertical do lado esquerdo. As áreas cor-de-rosa e amarela representam os tempos passados no *frontend* e *backend*. Os valores médios do tempo de carregamento são representados pela linha. O eixo dos valores das três métricas está do lado direito. A área que representa o *backend*, a cor-de-rosa, é pouco visível porque os seus valores são muito reduzidos comparados com os do *frontend*.

Os utilizadores navegam na Web em condições diferentes. Utilizam sistemas operativos, *browsers* e dispositivos diferentes, e acedem à Web a partir de diversas localizações. Todas estas variáveis podem contribuir para que o desempenho das páginas Web varie. Encontrar um padrão nos dados de desempenho dos utilizadores pode ajudar a identificar os problemas de desempenho. Por exemplo, pode-se concluir que apenas um segmento de utilizadores que acede a partir de uma localização geográfica tem tempos de carregamento não aceitáveis. Os dados de desempenho podem ser filtrados por página Web, tipo de dispositivo, país, *browser* e sistema operativo.

	Requests	Backend	Frontend	Load Time
Por recolher	94	0.56s	7.24s	7.87s
Russia	2	0.00s	67.63s	67.63s
France	1	0.38s	1.70s	2.08s
Republic of Moldova	7	0.14s	1.25s	1.39s
Portugal	55	0.34s	7.11s	7.55s
United States	289	0.54s	5.45s	6.00s
Todos	448	0.51s	6.24s	6.78s

Figura 131: Filtragem de dados de desempenho

A secção representada na Figura 131 permite visualizar os valores das métricas (*backend*, *frontend* e tempo de carregamento) em grupos de utilizadores com diversas condições. Do lado esquerdo está um gráfico que indica a percentagem de amostras com determinada condição, por exemplo na Figura 131 visualiza-se que 12.3% das amostras analisadas foram feitas a partir de Portugal. Do lado direito está uma tabela com os diferentes países a partir dos quais os utilizadores acederam, e o número de amostras e tempos médios das três métricas associados. Ao selecionar um dos países presentes na tabela, os dados visualizados na página são filtrados, sendo visualizados dados das amostras que tiveram acesso a partir desse país. O mesmo funcionamento se aplica às outras variáveis selecionáveis (página Web, tipo de dispositivo, sistema operativo e *browser*).

7.4 Otimização de imagens

Enquanto se realizavam os redimensionamentos das imagens, cujas dimensões não eram as apresentadas nas páginas Web, sentiu-se a necessidade de ter uma ferramenta capaz de aceder a elas e gerar as várias imagens com os tamanhos corretos de uma só vez. Uma ferramenta com estes requisitos ajudaria a aumentar a produtividade na otimização das imagens, evitando que o tratamento tivesse que ser feito uma a uma. Antes da criação da página com esta funcionalidade, encontraram-se soluções na Web, das quais se tiraram ideias para criar a página a apresentar.

Uma plataforma Web encontrada oferece um conjunto de serviços que podem ser utilizados por qualquer pessoa. Eles são:

- **Image-resize** (http://img-resize.com), onde podem ser ajustados os tamanhos das imagens.
- **CSS minifier** (http://cssminifier.com), extrai os espaços vazios de um ficheiro CSS reduzindo o seu tamanho.
- **Javascript minifier** (http://javascript-minifier.com), executa o mesmo procedimento que o "CSS minifier", mas em ficheiros javascript.

- **Png Crush** (http://pngcrush.com), uma ferramenta *open source* que utiliza métodos de compressão de imagens PNG e retira alguma informação inútil para a sua visualização na Web.
- **Jpeg Optimizer** (http://jpgoptimiser.com), à semelhança do "Png Crush" utiliza algoritmos de compressão de imagens mas com extensão JPEG.

Além da utilização dos serviços na plataforma, eles podem ser integrados com outras aplicações Web. Por exemplo, solicitar um pedido com o método POST, com uma imagem PNG para o URL http://pngcrush.com/crush, devolve como resposta a imagem otimizada. Na Figura 132 está representado o comando, com o programa "curl", que permite realizar o pedido HTTP POST referido.

```
curl -X POST -s --form "input=@filename.png;type=image/png" http://pngcrush.com/crush > crushed.png
```

Figura 132: POST para o png crush

Todas as ferramentas mencionadas seriam uteis na plataforma desenvolvida, mas uma vez que não se sentiu a necessidade de as utilizar para melhorar a aplicação Liberopinion, decidiu-se criar apenas a página com a funcionalidade descrita no início da secção.

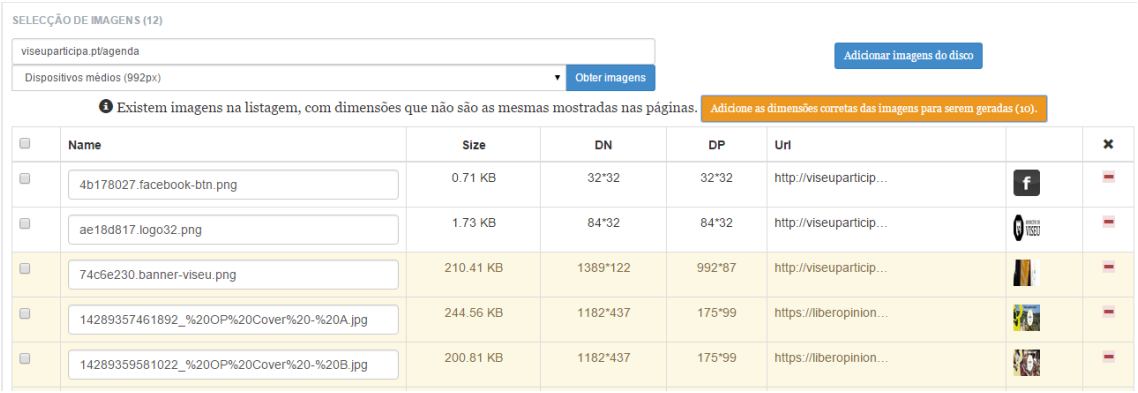


Figura 133: Formulário de seleção de imagens

Na Figura 133 está representado o formulário de seleção de imagens. No canto superior direito existe um botão através do qual se selecionam imagens que se encontram no disco. No canto superior esquerdo há um campo de texto, para indicar o URL da página de onde se extraem as imagens. É fornecida ainda a opção de escolher o tamanho da janela de visualização, para simular o acesso à página com dispositivos de dimensões inferiores. Esta funcionalidade permite analisar o tamanho das imagens nas várias experiências de utilização (Desktop, tablet ou smartphone).

As imagens recolhidas, selecionando um URL, provocam a execução de um script desenvolvido em Phantomjs, no servidor. Neste script é executado um pedido HTTP e é analisada a resposta, procurando imagens e informação sobre as suas dimensões.

Na Figura 133 são apresentadas imagens extraídas de http://viseuparticipa.pt/agenda. A página selecionada contém 12 imagens, e 10 delas não estão presentes na página com as suas dimensões naturais. Na tabela é exibido o nome, o tamanho, as dimensões naturais (DN), as dimensões na página (DP), a localização na Web e a miniatura de cada uma das imagens. As

linhas amarelas da tabela evidenciam as imagens com dimensões diferentes das visualizadas na página.

A partir da listagem apresentada, devem ser selecionadas as imagens e as dimensões para o seu redimensionamento. No topo da tabela, como visualizado na Figura 133, há um botão que seleciona as imagens desajustadas da página, e atribui automaticamente as dimensões que deveriam ter na página. Outra forma mais trabalhosa passa por selecionar na tabela as imagens com as mesmas dimensões, e depois preencher o formulário da Figura 134.

Figura 134: Formulário para selecionar dimensões de imagens a gerar

No formulário da Figura 134 estão os nomes das imagens selecionadas, a proporção da largura em relação à altura, a largura e a altura. Ao definir uma proporção, quando alterado o campo de largura ou altura, há um ajuste automático dos seus valores. Após selecionar as imagens e respetivas dimensões, elas são adicionadas à tabela presente na Figura 135.

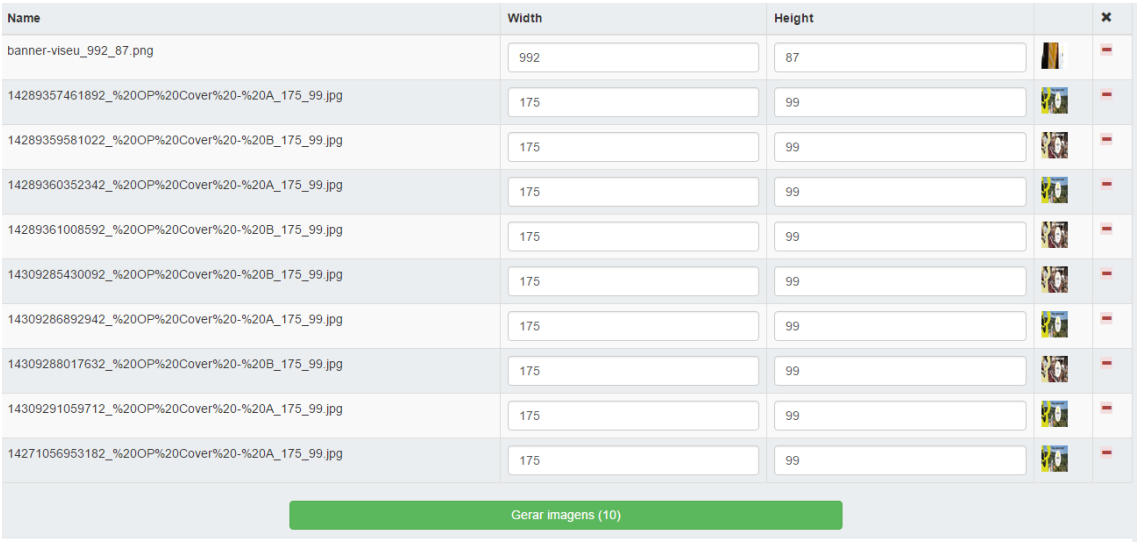
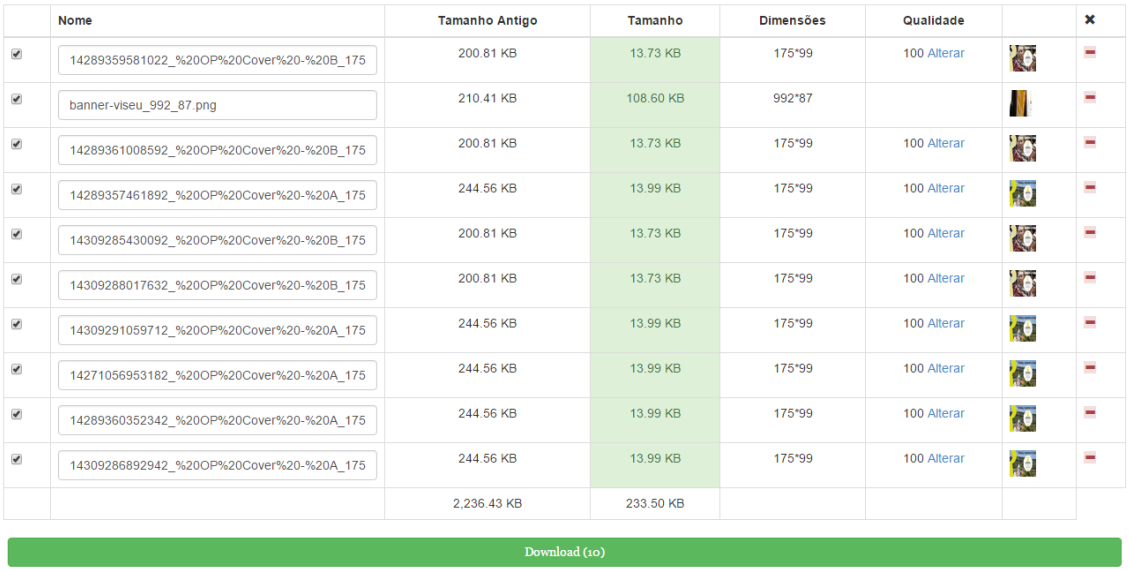


Figura 135: Tabela com imagens e dimensões selecionadas

Na tabela com as imagens a gerar, existe a possibilidade de alterar as suas dimensões. Selecionadas as imagens e as novas dimensões, clica-se no botão verde abaixo da tabela para gerar as imagens com as dimensões escolhidas. Os detalhes das imagens geradas são adicionados à tabela representada na Figura 136.



	Nome	Tamanho Antigo	Tamanho	Dimensões	Qualidade		✖
☑	14289359581022_%20OP%20Cover%20-%20B_175	200.81 KB	13.73 KB	175*99	100 Alterar		
☑	banner-viseu_992_87.png	210.41 KB	108.60 KB	992*87			
☑	14289361008592_%20OP%20Cover%20-%20B_175	200.81 KB	13.73 KB	175*99	100 Alterar		
☑	14289357461892_%20OP%20Cover%20-%20A_175	244.56 KB	13.99 KB	175*99	100 Alterar		
☑	14309285430092_%20OP%20Cover%20-%20B_175	200.81 KB	13.73 KB	175*99	100 Alterar		
☑	14309288017632_%20OP%20Cover%20-%20B_175	200.81 KB	13.73 KB	175*99	100 Alterar		
☑	14309291059712_%20OP%20Cover%20-%20A_175	244.56 KB	13.99 KB	175*99	100 Alterar		
☑	14271056953182_%20OP%20Cover%20-%20A_175	244.56 KB	13.99 KB	175*99	100 Alterar		
☑	14289360352342_%20OP%20Cover%20-%20A_175	244.56 KB	13.99 KB	175*99	100 Alterar		
☑	14309286892942_%20OP%20Cover%20-%20A_175	244.56 KB	13.99 KB	175*99	100 Alterar		
		2,236.43 KB	233.50 KB				

Download (10)

Figura 136: Imagens geradas

Em cada linha da imagem está representada a informação do tamanho antes do redimensionamento e os ganhos ou as perdas do seu tamanho em bytes. A utilização das imagens geradas na página de onde foram extraídas, reduz o seu tamanho em aproximadamente 233 Kilobytes (menos 10%). O tamanho pode ainda ser reduzido, diminuindo a sua qualidade, isto é comprimindo-a. Na coluna da "Qualidade", clicando na hiperligação surge a interface representada na Figura 137, que serve para tratar a qualidade da imagem.

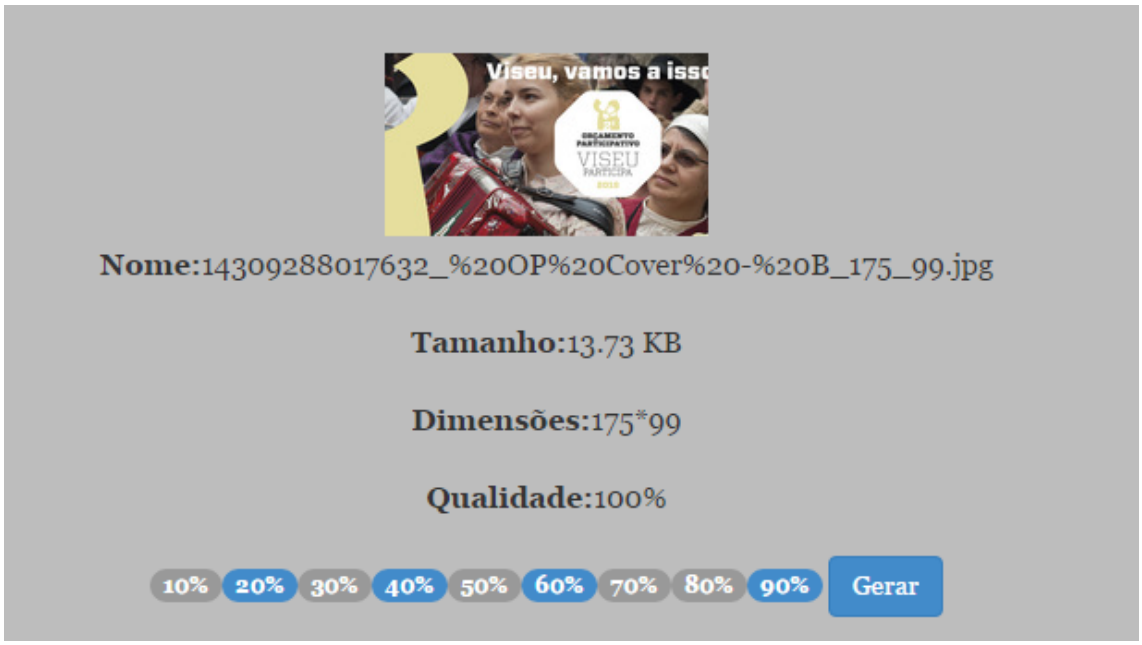


Figura 137: Interface para selecionar qualidade das imagens

Na Figura 137 visualizam-se a imagem e alguns dos seus detalhes. No fundo da figura selecionam-se as várias qualidades em que serão geradas as imagens. São apresentadas as imagens com as várias qualidades escolhidas para facilitar a comparação e a sua seleção. Clicando no botão "Gerar" as imagens nas diferentes qualidades são reveladas como representado na Figura 138.

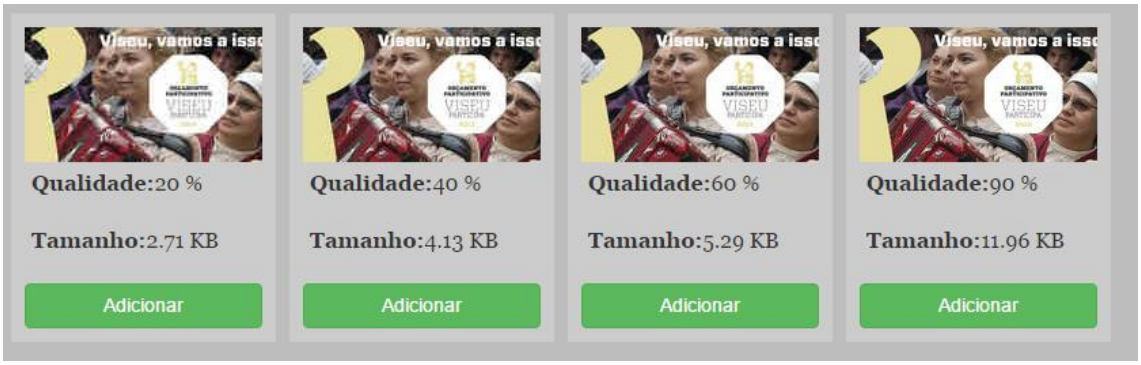


Figura 138: Imagem em várias qualidades

As imagens são apresentadas por ordem crescente da sua qualidade. Verifica-se que a qualidade da imagem repercute-se no seu tamanho. Verifica-se que a imagem mais à esquerda apresenta mais deformações, pois notam-se mais os seus pixéis. Na escolha da imagem deve-se balancear o tamanho e a qualidade. Ao clicar no botão "Adicionar" a imagem é adicionada à lista de imagens a transferir.

Clicando no botão de *download*, por baixo da tabela das imagens geradas, na Figura 136, estas são transferidas para uma pasta da máquina com as modificações selecionadas nos formulários apresentados. A ferramenta utilizada no servidor para fazer o tratamento de imagens (redimensionamentos e ajuste de qualidade) é o "ImageMagick".

8 Apresentação e análise de resultados

O conhecimento de formas de medição de desempenho e técnicas para melhorar os tempos de resposta de uma aplicação Web auxiliam a execução de um dos principais objetivos deste trabalho, a melhoria de desempenho de uma aplicação Web.

Sem medição não é possível tirar conclusões sobre o impacto das modificações feitas para melhorar os tempos de resposta das páginas Web. Enquanto se implementam as técnicas de melhoria deve-se determinar objetivamente o valor das métricas de performance a atingir. Antes de estipular valores analisam-se criticamente as métricas a melhorar.

8.1 Métricas e objetivos de performance

Ao medir o desempenho de uma aplicação existe a tendência de procurar melhorar o tempo de carregamento da página Web. Este indica o tempo passado desde o início da navegação até ao momento em que são transferidos todos os recursos da página HTML. Uma abordagem diferente para determinar que uma página Web está pronta a ser utilizada, é o tempo de carregamento total, medido na plataforma Webpagetest, que é dado pelo início da navegação até ao instante em que cessam os pedidos de rede. Estas duas métricas não espelham o tempo verdadeiro que os utilizadores estão num estado de espera porque quando a página está completamente renderizada, o utilizador deixa de estar em espera para iniciar as suas tarefas, independentemente de ainda haver recursos a serem transferidos.

Os dois momentos que afetam a perceção dos utilizadores são o início e o fim da renderização. O início da renderização indica que a página existe e brevemente estará disponível para ser utilizada. Um elevado tempo de espera para o início da renderização pode influenciar o utilizador a desistir de aceder a uma página, levando-o a pensar que ocorreu algum problema

no sistema ou que não se encontra disponível no momento. Uma métrica apresentada que indica o fim da renderização é o *Speed Index.* Esta revela o tempo que demoram a ser renderizados os elementos na janela de visualização. Embora a renderização possa não estar completa no resto da página não visível, o utilizador tem a perceção que a página está pronta a ser utilizada quando deixam de aparecer elementos gráficos novos na sua janela de visualização.

Segundo Jakob Nielsen [8], o ideal é a página estar disponível em 100 milissegundos, este é o tempo limite para o utilizador ter a sensação que o sistema responde instantaneamente às suas ações. A ilusão de agir diretamente sobre o sistema provoca no utilizador um bem-estar que maximiza a produtividade na execução das suas tarefas, como explica o psicólogo Mihaly Csikszentmihalyi, na teoria do *Flow* [2].

Os 100 milissegundos são um limite muito ambicioso, tendo em conta a quantidade de processos envolvidos na disponibilização de uma página Web. Uma meta mais realista, mas extremamente ambiciosa também, é a estabelecida pela Google, uma página Web deve ser carregada num segundo, independentemente do dispositivo e o tipo de rede utilizado para lhe aceder [39]. Como se verificou na apresentação das pesquisas de Jakob Nielsen, um segundo é o tempo limite para não interromper a corrente de pensamento do utilizador, embora ele se aperceba do atraso [8]. A métrica que deve apresentar este valor não é o tempo de carregamento, mas o tempo que leva a ser renderizado o conteúdo presente na janela de visualização, ou seja o valor do *Speed Index*. As características da ligação de redes móveis 3G revelam que há muito pouco espaço de manobra na otimização das páginas Web nos dispositivos móveis para a página ser renderizada num segundo [39].

Um questionário levado a cabo pela empresa Akamai [40], refere que 47% dos consumidores inquiridos, esperam que uma página esteja acessível em 2 segundos ou menos. O tempo de espera a que os utilizadores estão sujeitos enquanto navegam na Web, torna-os tolerantes a tempos de espera superiores.

Os estudos estatísticos elaborados por Tammy Everts [40] revelam que o tipo de conteúdo presente nas páginas Web contribui para que os utilizadores tolerem diferentes tempos de espera [40]. As características dos utilizadores também são um fator que influenciam a tolerância, um estudo revela que os utilizadores australianos são mais pacientes que os norte-americanos [40]. Dada a diversidade de variáveis que influenciam o comportamento humano, cada caso é um caso, e o estabelecimento das metas de melhoria devem basear-se em dados estatísticos reais da aplicação a melhorar.

8.2 Método de referência para a otimização de performance

Tammy Everts [40] sugere seguir um método para estabelecer as metas dos valores das métricas a atingir. O método é composto pela medição, análise de resultados e monitorização.

A aplicação desenvolvida para auxiliar a medição de dados de performance na aplicação teste está preparada para recolher dados de performance de utilizadores reais de qualquer *website*. Cada vez que um utilizador acede a uma página Web monitorizada, é transferido um script que recolhe os valores das métricas de performance e os envia para o servidor da

aplicação desenvolvida. Os dados de performance recolhidos são o tempo de procura DNS, o tempo de estabelecimento de ligação TCP, o tempo de resposta do servidor, o tempo de processamento da página, o tempo de carregamento e o número de recursos transferidos.

A aplicação teste trata-se de uma *single page application.* Durante a navegação neste tipo de aplicações, o evento do *browser* que indica que a página está pronta a ser utilizada apenas é disparado uma vez. O evento em causa é utilizado para iniciar a recolha de dados de performance. A *framework* javascript utilizada, altera assincronamente o conteúdo da página, simulando uma mudança de página que não é detetada pelo *browser*. As características da aplicação levam a que recolha de dados de desempenho apenas seja feita no primeiro acesso à aplicação.

O primeiro acesso à aplicação é o mais impactante para a experiência do utilizador. A primeira vez que se acede à aplicação, são transferidos todos os recursos da sua estrutura, isto é, os ficheiros HTML, CSS e javascript. Em acessos subsequentes e mudanças de página, parte dos recursos estão guardados e carregados localmente evitando-se assim pedidos de rede, o que torna a "mudança" de página mais rápida. Em termos visuais, a experiência é mais fluida para o utilizador porque parte da página renderizada mantem-se. Nas aplicações tradicionais, a mudança de página implica que a página visualizada desapareça e depois fique vazia até a nova página ser renderizada, o que afeta a experiência de utilização. Considerou-se que a recolha de dados do primeiro acesso era suficiente para analisar o comportamento dos utilizadores.

A seguir à medição analisam-se os dados, relacionando a utilização da plataforma com a performance, de forma a identificar os valores objetivo das métricas de performance. No caso de um *website* de comércio online, seriam utilizadas métricas como número de aquisições ou lucro obtido. No caso estudado, as métricas referidas não se aplicam e métricas como a taxa de rejeição, o número de visitas e o tempo médio na aplicação têm mais sentido. O Google Analytics e a plataforma desenvolvida são ferramentas valiosas para analisar os dados.

Os valores de performance variam ao longo do tempo, à medida que há atualizações na plataforma, portanto é importante haver uma monitorização dos valores das métricas para não ultrapassarem as metas estabelecidas. Há sempre aspetos que podem melhorar o desempenho de uma página Web, mas segundo Tammy Everts, devem ser resolvidos os problemas que afetam mais gravemente o desempenho, pois não há tempo e dinheiro para otimizar até à exaustão cada um dos aspetos de performance.

Ao estabelecer os valores objetivo para as métricas de performance, inicia-se a otimização da aplicação teste. A análise do impacto das alterações efetuadas através de dados recolhidos dos utilizadores reais, requer tempo para recolher uma amostragem capaz de evidenciar os resultados da implementação. Além disso, as diferentes condições de acesso dos utilizadores fazem com que os valores das métricas de performance variem substancialmente, induzindo em conclusões erradas sobre o impacto das técnicas de melhoria implementadas. Portanto, a solução da medição do impacto passa por utilizar testes sintéticos, onde as condições de acesso são controladas.

Em suma, o processo de melhoria da otimização passa pelos seguintes passos.

1. Recolher dados de performance e dados de utilização da plataforma num cenário controlado de utilização;

2. Analisar os dados de performance com os dados de utilização da plataforma, de forma a tirar conclusões sobre que páginas melhorar e definir valores aceitáveis nas métricas de performance da página;
3. Implementar melhorias de otimização e realizar testes sintéticos até obter resultados aceitáveis ou esgotarem as soluções de otimização.

8.3 Análise de resultados de performance

A monitorização de utilizadores reais é importante, uma vez que os seus dados espelham a experiência de utilização da aplicação Web. A diversidade das condições em que são feitos os acessos permite identificar problemas que não seriam detetáveis em ambientes controlados, como os testes sintéticos.

A Liberopinion é uma plataforma personalizável que é utilizada em vários *websites*. Os dados de performance foram recolhidos dos *websites*, através da aplicação desenvolvida, a partir das datas mencionadas a seguir.

- http://covilhadecide.pt (desde 3 de Junho de 2015)
- http://pombalparticipa.pt (desde 9 de Junho de 2015)
- http://opj.cmhorta.pt (desde 12 de Junho de 2015)
- http://op.cm-albergaria.pt (desde 25 de Junho de 2015)
- http://op.cm-batalha.pt (desde 26 de Junho de 2015)
- http://orcamentoparticipativo.cm-lourinha.pt (desde 9 de Junho de 2015)
- http://viseuparticipa.pt (desde 17 de Julho de 2015)

Na análise da performance das várias aplicações verificou-se que os dados recolhidos pela plataforma não eram suficientes. Havia necessidade de comparar os dados de performance com os dados de utilização da plataforma, e estes últimos não eram recolhidos.

Ao iniciar a recolha de dados com a plataforma desenvolvida, utilizou-se também o Google Analytics para recolher dados de performance, por questão de controlo de resultados.

São apresentados a seguir os dados de utilizadores reais sobre o número de acessos e o tempo de carregamento de página por dia, em forma de gráfico, entre o dia 28 de Agosto e 27 de Setembro, no *website* do Orçamento Participativo da Lourinhã (http://orcamentoparticipativo.cm-lourinha.pt). Na Figura 140 são apresentados os dados recolhidos da plataforma desenvolvida e na Figura 140 os dados recolhidos do Google Analytics.

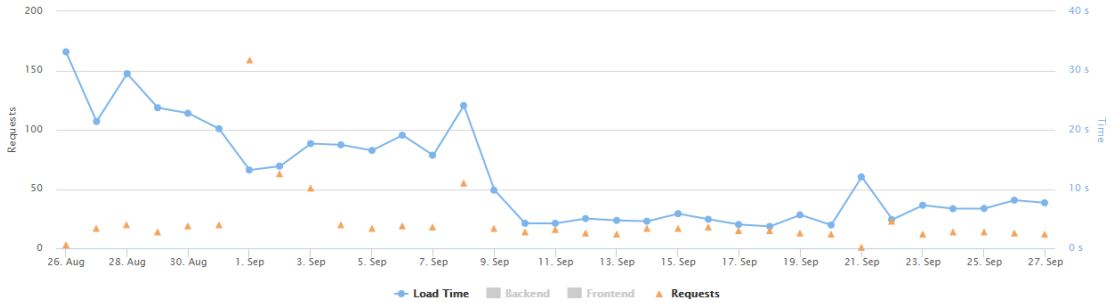


Figura 139: Dados de performance recolhidos pela plataforma desenvolvida

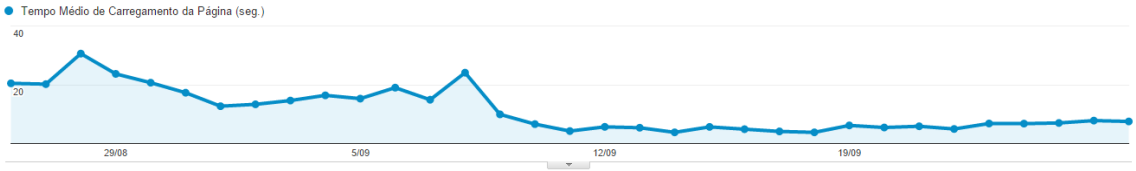


Figura 140: Dados de performance recolhidos pelo Google Analytics

Na Figura 141 são comparados os dados recolhidos pelas duas plataformas. Verifica-se que os resultados são muito semelhantes, com exceção nos dias 26 de Agosto e 21 de Setembro. Uma vez que nos restantes dias, os valores de tempo de carregamento praticamente coincidem, o processo de recolha de dados não é muito diferente. A única explicação pela discrepância nos dois dias mencionados, reside na falha de alguma das partes em recolher o tempo de carregamento em determinadas sessões.

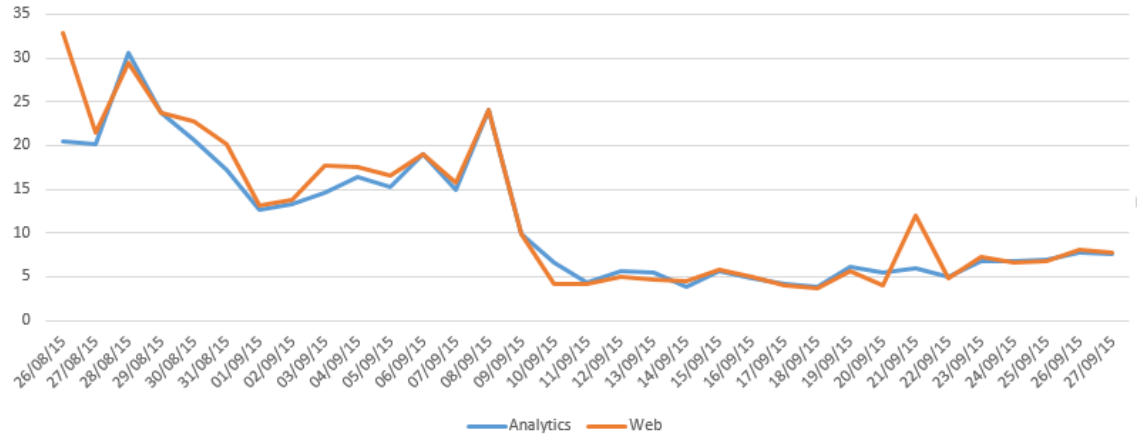


Figura 141: Dados recolhidos pelo Google Analytics e pela plataforma Web

Uma vez que a plataforma desenvolvida não está preparada para analisar o comportamento dos utilizadores, chegou-se à conclusão que a análise da performance no Google Analytics seria a opção acertada.

A métrica que permite visualizar o impacto do tempo de carregamento na saída da aplicação é a taxa de sessões rejeitadas. Uma sessão rejeitada diz respeito a uma sessão onde o utilizador sai do *website* após visualizar a primeira página Web. Esta variável foi utilizada também nos estudos estatísticos apresentados anteriormente. Uma das conclusões retiradas na análise dos estudos estatísticos, foi que a maioria dos compradores *online* abandonavam um

website se demorasse mais de 3 segundos a ficar disponível. Começou-se por comparar o tempo médio de carregamento com a taxa de rejeição por dia através do gráfico da Figura 142.

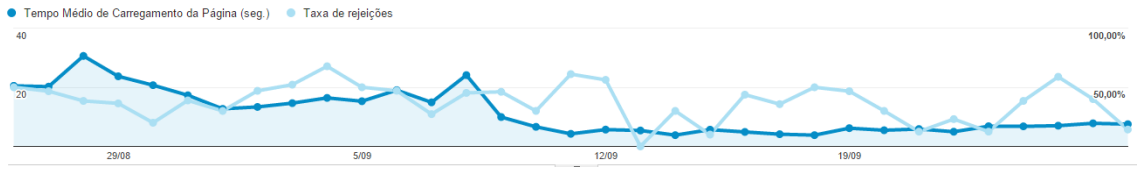


Figura 142: Tempo médio de carregamento e taxa de rejeição por dia

Verifica-se que a linha azul-escuro, que representa o tempo médio de carregamento, tem valores mais baixos com o decorrer do tempo mas a taxa de rejeição, representada pela linha azul claro, é inconstante, o que impossibilita tirar qualquer conclusão.

O método anterior não forneceu informações revelantes, portanto decidiu-se analisar as taxas de rejeição, filtrando as amostras de acordo com o seu tempo de carregamento de página. Foram analisadas as amostras recolhidas entre os dias 27 de Agosto de 2015 e 26 de Setembro de 2015 em sete *websites*, como representado na Tabela 3.

Website	Total de amostras	Sessões rejeitadas	Utilizadores repetidos
Batalha	448	26	48
Pombal	1789	334	729
Albergaria	830	146	204
Horta	435	32	42
Lourinhã	771	126	177
Covilhã	20373	3592	12313
Viseu	8933	2439	2671

Tabela 3: Número de amostras analisadas

Filtraram-se as amostras por intervalos de tempo de carregamento e criou-se um gráfico de barras onde estão representadas as taxas de rejeição.

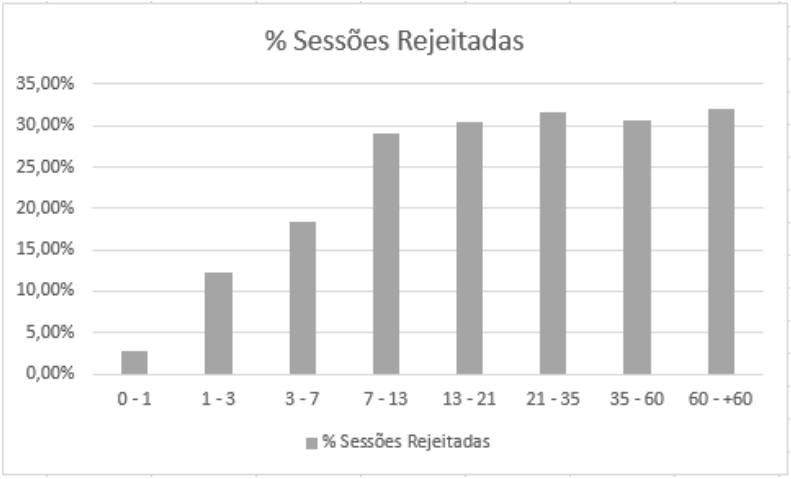


Figura 143: Sessões rejeitadas

Na Figura 143, está representado um gráfico com as taxas de rejeição em diferentes intervalos de tempo de carregamento. No número total de sessões onde o tempo de

carregamento de página demorou entre 0 e o 1º segundo, apenas em aproximadamente 2,5% das sessões, os utilizadores saíram do *website* sem visitar outra página. Demonstra-se que a taxa de rejeição tende em crescer à medida que aumenta o tempo do carregamento das páginas Web. Para um tempo de espera de carregamento superior a sete segundos, a taxa de rejeição manteve-se constante, com cerca de 30% das sessões a serem rejeitadas.

Conclui-se assim que o tempo de carregamento tem impacto em aplicações de participação pública. A taxa de rejeição estabiliza para tempos de carregamento de página superiores a sete segundos, por isso quanto mais sessões tiverem tempos de carregamento inferiores a sete segundos, menos rejeição há na utilização da plataforma.

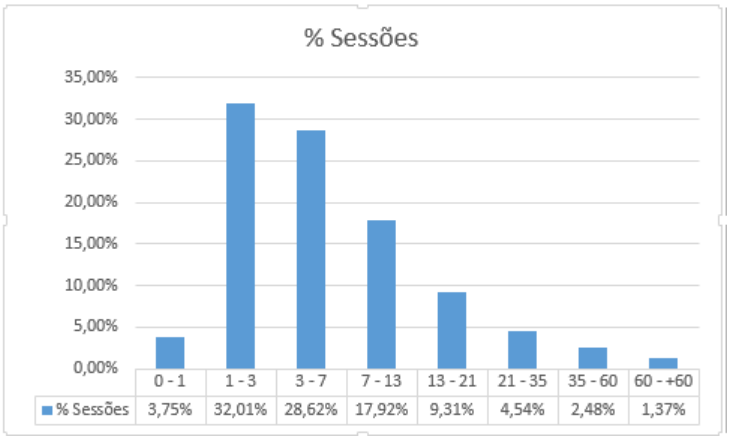


Figura 144: Percentagem de amostra por intervalo de tempo de carregamento

No gráfico da Figura 144, verifica-se que 64% das sessões já têm tempos de carregamento inferiores a sete segundos. As condições de acesso dos utilizadores são muito diferentes e consequentemente os tempos de carregamento também. É impossível garantir que todos os utilizadores tenham um tempo de carregamento inferior a sete segundos, mas quanto melhor for o desempenho da aplicação, maior é a probabilidade de tal acontecer.

A métrica de desempenho analisada foi o tempo de carregamento. Referiu-se que as métricas que melhor ilustram a experiência de utilização são o início da renderização e o *Speed Index*, mas os seus valores não são recolhidos na monitorização dos utilizadores reais. Dá-se maior destaque ao tempo de carregamento, na implementação das técnicas de melhoria, porque é a métrica recolhida dos utilizadores para estudar o seu comportamento.

8.4 Otimização da aplicação e resultados

Os testes sintéticos são executados em ambientes controlados e dão *feedback* sobre o desempenho da aplicação quando acabam de ser executados. Neles não há o fator da variabilidade da rede a influenciar os tempos de resposta, portanto os resultados são estáveis de teste para teste, o que permite medir o impacto das melhorias implementadas.

São descritas a seguir as técnicas implementadas e os resultados de performance dos testes sintéticos executados a partir do WebpageTest. Os testes são feitos à página inicial da aplicação Liberopinion porque o primeiro acesso é feito com mais frequência através dela. A Figura 145

ilustra o fluxo dos utilizadores num dos *websites* onde se recolhem dados. No caso ilustrado, de 1300 sessões, 724 (55.7%) foram iniciadas na página inicial da plataforma (/home). Nos outros websites segue-se a mesma tendência, com exceção ao *website* http://viseuparticipa.pt, cuja página mais acedida no primeiro acesso é a página da listagem dos projetos.

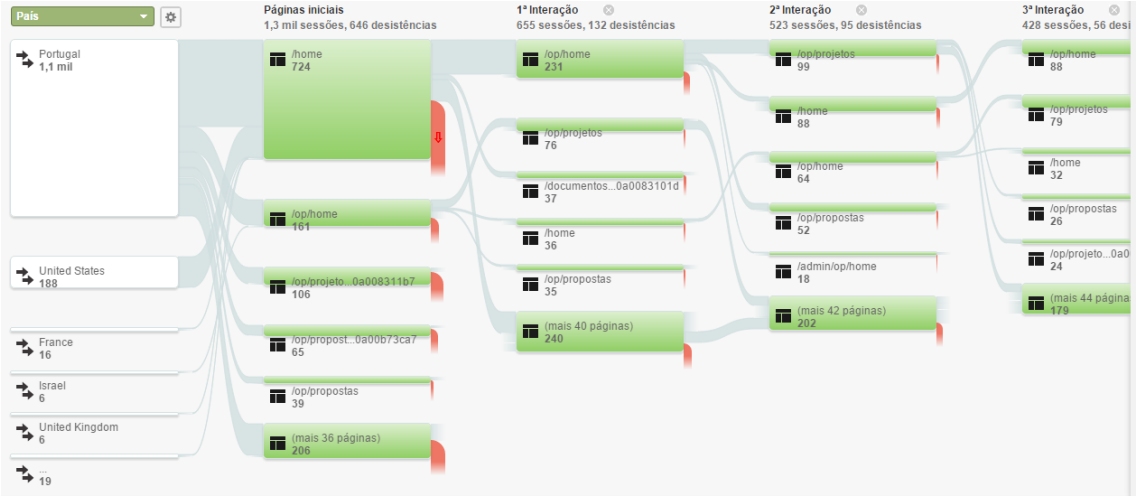


Figura 145: Fluxo dos utilizadores

Contudo as melhorias aplicam-se às restantes páginas da plataforma. Tratando-se de uma *single page application*, são transferidos recursos em comum em todas as páginas no primeiro acesso, como é referido anteriormente.

Segundo dados recolhidos pela Akamai [41], em 2015, a largura de banda média global é de 5 Megabits por segundo. As configurações escolhidas para os testes sintéticos foram as seguintes.

- **Localização**: Dulles, EUA;
- ***Browser*:** Firefox;
- **Repetição do teste:** 5 vezes;
- **Conexão**: Cabo, com 5 Mbps de *download* e 1 Mbps de *upload;*
- Visualização com *cache* cheia e vazia;

A ordem das técnicas de melhoria implementadas está relacionada com o impacto que elas têm na aplicação, isto é, a dificuldade de integração de código ou ferramentas para implementar as técnicas.

8.4.1 Sem técnicas de melhoria de performance

Inicialmente já estavam implementadas na aplicação algumas técnicas de melhoria apresentadas. Os recursos javascript encontravam-se no fundo da página HTML e os recursos CSS no topo. Já existia uma política de armazenamento em *cache*, tanto para os recursos da aplicação como para as imagens, embora não existissem tempos de expiração. Os cabeçalhos relativos à política de armazenamento em *cache* eram "Cache-Control: public, max-age=0". Assim os vários ficheiros eram armazenados em *cache*, mas eram feitos sempre pedidos de rede

ao servidor para verificar se eles não tinham sido alterados. A transferência dos recursos apenas era feita quando a versão do ficheiro armazenado em *cache* era diferente da presente no servidor.

Realizou-se um teste sintético antes da implementação de novas técnicas melhorias, e obteve-se os resultados presentes na Figura 146.

						Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 4)	36.309s	0.257s	23.838s	26332	559	36.309s	371	10,812 KB	36.689s	376	10,834 KB	$$$$$
Repeat View (Run 2)	7.672s	0.229s	7.902s	8500	337	7.672s	327	276 KB	9.538s	359	392 KB	

Figura 146: Resultados sem técnicas de melhoria de performance

As linhas da tabela da Figura 146 dizem respeito aos valores das métricas em dois cenários. Um é a primeira visualização da página Web e noutro é a visualização repetida, onde já existem recursos armazenados em *cache*. As métricas apresentadas são o tempo de carregamento (*Load Time*), o tempo de resposta do servidor depois de estabelecida a conexão TCP (*First Byte*), o início da renderização (*Start Render*), o *Speed Index* e o número de elementos da página HTML (*Dom Elements*).

Existem métricas agrupadas nas secções "*Document Complete"* e "*Fully Loaded"*. As duas secções contêm valores de métricas que ocorrem em dois momentos diferentes do carregamento de página. O *Document Complete* indica os valores contabilizados quando são transferidos todos os recursos presentes na página HTML. Depois deste momento, pode haver transferência de mais conteúdo assíncrono. O *Fully Loaded* ocorre quando cessam os pedidos de rede. Durante a apresentação dos resultados, o **tempo de carregamento** diz respeito à métrica tempo do "*Document Complete*" e o **tempo de carregamento total** ao tempo do "*Fully Loaded*".

Os resultados dos testes realizados demonstram que em média, nos 5 testes realizados, no primeiro acesso à aplicação, o tempo de carregamento da página é de 36.3 segundos e a renderização inicia-se aos 23.8 segundos. O *Speed Index* é expresso em milissegundos, portanto a página encontra-se totalmente renderizada, na janela de visualização do utilizador aos 26.3 segundos.

De acordo com o estudo estatístico realizado, a partir dos 7 segundos de tempo de carregamento, a taxa de rejeição mantem-se nos 30%. Até aos 23.8 segundos não é renderizado qualquer elemento da página. Segundo Jakob Nielsen, a partir dos 10 segundos os utilizadores têm necessidade de realizar outras tarefas enquanto esperam, portanto há grandes probabilidades de o utilizador desistir de aceder à aplicação. Os valores de desempenho da aplicação neste estado estão longe de ser resultados satisfatórios para fornecer uma boa experiência de utilização na primeira visualização.

Na visualização repetida, o tempo de carregamento da página é de 7.6 segundos, o início da renderização é de 7.9 segundos e encontra-se renderizada aos 8.5 segundos. Embora os valores sejam mais próximos do desejável, ainda são superiores a 7 segundos. A visualização neste cenário acontece apenas quando ocorre a primeira visualização, portanto é insuficiente melhorar os valores das métricas apenas neste caso.

Os resultados deste teste servem como base para comparar os valores dos cenários onde são aplicadas as técnicas de melhoria e verificar o seu impacto na aplicação.

8.4.2 Gzip

						Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 5)	23.473s	0.240s	23.666s	27323	337	23.473s	335	1,597 KB	36.160s	376	7,514 KB	$$$$$
Repeat View (Run 5)	7.227s	0.237s	7.691s	8564	337	7.227s	326	280 KB	9.300s	360	341 KB	

Figura 147: Resultados com a utilização de Gzip

Na tabela da Figura 147, na primeira visualização de página, a utilização de Gzip apresenta melhorias significativas no tamanho do conteúdo transferido.

Primeira Visualização:

- **tempo de carregamento –** 23.4 segundos (menos 35%);
- **bytes transferidos** – 7,514 Kilobytes (menos 31%).

Os valores do tempo de carregamento total, do início da renderização e do *Speed Index* mantêm-se aproximadamente iguais aos valores do cenário sem melhorias. As melhorias do cenário da visualização repetida não são significativas.

Segundo o *browser* a página é carregada mais rapidamente, aproximadamente no momento em que é iniciada a renderização. Tal facto é transparente para o utilizador, portanto não afeta a sua experiência de utilização. Na Figura 148, são representados parcialmente os gráficos em cascata dos dois cenários, o da direita diz respeito ao cenário onde é utilizado Gzip.

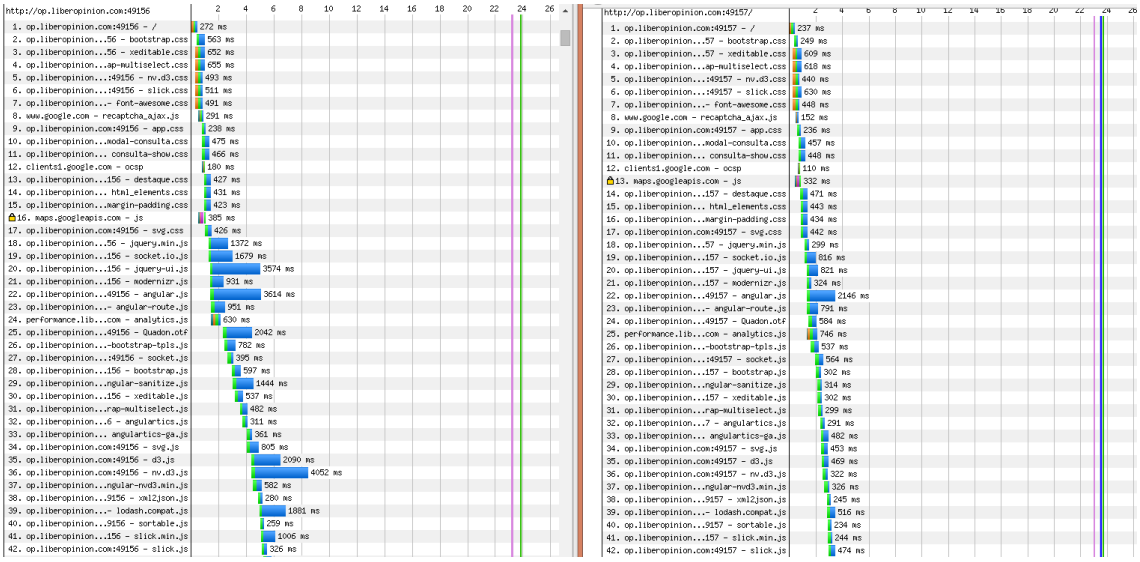


Figura 148: Comparação de gráficos em cascata dos cenários com Gzip e sem

Na Figura 148 visualiza-se que no cenário com Gzip, o tempo passado na transferência de recursos é menor. Esse tempo é representado nos gráficos pelas barras horizontais azuis, que no gráfico da esquerda têm maior dimensão. A capacidade que o *browser* tem de transferir os recursos em paralelo, leva a que a redução do tempo de transferência não influencie o tempo de

carregamento da página. A grande quantidade de recursos (376) tem um maior impacto na performance da página do que o seu tamanho. Os ficheiros javascript são transferidos e executados mais rapidamente, fazendo disparar mais cedo o evento javascript *onLoad*, o que indica que os recursos da página foram todos transferidos.

A implementação da melhoria consistiu em configuração o servidor, adicionando a linha de código da Figura 149.

```
app.use(express.compress());
```

Figura 149: Configuração de Gzip no servidor node

A linguagem utilizada no servidor é o Node e utiliza-se a *framework* Expressjs para construir a aplicação. A linha de código adicionada faz com que todas as respostas do servidor sejam comprimidas com o método de compressão Gzip.

8.4.3 Combinação de recursos e ofuscação do Javascript

A aplicação em causa está estruturada com uma *framework* javascript denominada Angular, que torna o *website* numa *single page application*. A utilização dela, obriga a que a lógica da aplicação passe a estar toda do lado do cliente, sendo transferido grande parte dos seus recursos no primeiro acesso. A utilização do padrão de desenho *Lazy loading* permite que a transferência dos recursos apenas seja feita quando necessária, mas devido ao seu impacto na estrutura da aplicação, não foram feitas pesquisas para a sua implementação.

A modularização de uma aplicação torna o seu código mais legível e a sua manutenção mais fácil. Ela implica que haja uma estruturação que leva à criação de uma grande quantidade de ficheiros distribuídos por várias pastas. Na página avaliada são transferidos 376 recursos, entre eles estão ficheiros javascript e CSS, comuns a todas as páginas da plataforma, e os recursos próprios da página, como imagens e dados de pedidos assíncronos.

						Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 2)	19.435s	0.224s	7.218s	10142	250	19.435s	56	8,821 KB	19.754s	61	8,842 KB	$$$$$
Repeat View (Run 2)	2.664s	0.254s	1.667s	2341	246	2.664s	40	255 KB	3.040s	45	263 KB	

Figura 150: Resultados da combinação de ficheiros javascript e CSS

Os resultados dos testes sintéticos, revelados na Figura 150, após combinar os recursos do mesmo tipo, resultou na diminuição de 376 para 61 recursos. As melhorias são significativas tanto na primeira visualização como na visualização repetida.

Primeira Visualização:

- **tempo de carregamento –** 19.4 segundos (menos 46.6%);
- **início da renderização –** 7.2 segundos (menos 69.8%);
- ***speed index*** – 10142 (menos 62.5%);
- **tempo de carregamento total** – 19.7 segundos (menos 47.2%);
- **bytes transferidos** – 8842 Kilobytes (menos 18.2%).

Visualização repetida:

- **tempo de carregamento –** 2.7 segundos (menos 64.4%);
- **início da renderização –** 1.7 segundos (menos 78.5%);
- ***speed index*** – 2341 (menos 72.5%);
- **tempo de carregamento total** – 3 segundos (menos 68.4%);
- **bytes transferidos** – 255 Kilobytes (menos 7.6%).

A técnica de melhorou os tempos de renderização e de carregamento em taxas superiores a 50%. Além da combinação dos recursos, os ficheiros javascript foram ofuscados, o que contribuiu para uma redução de bytes transferidos de 18.2% na primeira visualização e 7.6% na visualização repetida. A ofuscação não reduziu ainda mais o tamanho dos bytes transferidos porque os templates HTML foram copiados para os ficheiros javascript. A utilização de um serviço do Angular, o "$templateCache", permite que o HTML seja carregado a partir do javascript. Ao carregar as páginas HTML, a partir do código javascript, são evitados pedidos de rede.

A combinação dos ficheiros do mesmo tipo, a ofuscação e a cópia dos *templates* HTML foram feitas com auxílio a *software* de automação. O desenvolvimento de uma aplicação Web com ficheiros minificados ou ofuscados é complexa, portanto deve haver uma versão do código da aplicação para desenvolver e outra para colocar em produção. Cada vez que é adicionado ou alterado um recurso javascript, CSS ou HTML, é necessário implementar as técnicas de melhoria abordadas, para passar o *website* para produção, o que se revela uma tarefa repetitiva e trabalhosa. Um *software* de automação facilita este tipo de tarefas. O software utilizado na implementação destas técnicas de melhoria foi o Grunt. A utilização do Grunt consiste na criação de um ficheiro javascript, na pasta da aplicação, com as instruções das tarefas a serem executadas, como apresentado na Figura 151.

```
grunt.registerTask('build', function(target){
  var tasks=[
    'clean:dist',
    'sass-config',
    'injector:sass',
    'concurrent:dist',
    'injector',
    'wiredep',
    'useminPrepare',
    'autoprefixer',
    'ngtemplates',
    'concat',
    'ngAnnotate',
    'copy:dist',
    'cdnify',
    'cssmin',
    'uglify',
    'rev',
    'eol',//http://stackoverflow.com/questions/23895397/g
    'usemin',
    'replace:processEnv'
  ];

  if(target){
    grunt.config.set('instancia',target);
    var config=require('./server/config/'+target+'.js');
    grunt.config.set('gaInstancia',config.gaID);
    tasks.splice(0,0,'replace:clientConfigInstancia');
    tasks.push('replace:serverConfigInstancia');
    tasks.push('replace:googleAnalytics');
  }

  return grunt.task.run(tasks);
});
```

Figura 151:Tarefa do Grunt

No código da Figura 151, é registada a tarefa “build”, que tem associada um outro conjunto de tarefas que reformulam o código da aplicação para ser colocado em produção, já com os recursos do mesmo tipo combinados e ofuscados. Na variável “tasks” estão as várias tarefas a serem executadas quando se pretende desempenhar a tarefa “build”, entre elas estão:

- **Ngtemplates,** que copia todas as páginas HTML para um ficheiro javascript.
- **Concat,** que combina todos os recursos javascript e CSS num recurso do seu tipo.
- **Uglify,** que ofusca os recursos javascript.

A cada uma das tarefas presentes no registo da tarefa “build”, está associada uma configuração que descreve como deve ser processada.

```
// Package all the html partials into a single javascript payload
ngtemplates: {
  options: {
    // This should be the name of your apps angular module
    module: 'appLibertrium',
    htmlmin: {
      collapseBooleanAttributes: true,
      collapseWhitespace: false,
      removeAttributeQuotes: true,
      removeEmptyAttributes: true,
      removeRedundantAttributes: true,
      removeScriptTypeAttributes: true,
      removeStyleLinkTypeAttributes: true
    },
    usemin: 'app/app.js'
  },
  main: {
    cwd: '<%= yeoman.client %>',
    src: ['{app,components}/**/*.html'],
    dest: '.tmp/templates.js'
  },
  tmp: {
    cwd: '.tmp',
    src: ['{app,components}/**/*.html'],
    dest: '.tmp/tmp-templates.js'
  }
},
```

Figura 152: Configuração da tarefa ngtemplates

Por exemplo, a tarefa configurada na Figura 152 é a que copia as páginas HTML para um ficheiro javascript. Na propriedade "src" é definido o caminho das páginas HTML que devem ser convertidas para um ficheiro javascript que se encontra no caminho indicado na propriedade "dist".

A construção da aplicação para produção, passa depois pela execução do comando, apresentado na Figura 153, na pasta onde está presente o ficheiro de configuração do Grunt.

```
Fabio@XYZ ~/www/liberopinion2 (performance)
$ grunt build
```

Figura 153: Comando de execução da tarefa do Grunt

O programador pode assim trabalhar numa versão da aplicação onde os ficheiros não estão combinados e ofuscados e cada vez que é necessário atualizar a aplicação num servidor, executar previamente o comando do *software* de automação.

8.4.4 *Cache*

A definição do tempo de expiração para armazenamento dos recursos guardados em *cache* foi feita seletivamente. As páginas HTML e os dados enviados pela API não devem ter um tempo de expiração.

Os dados são mutáveis ao longo do tempo, mantê-los em *cache* sem verificar se a resposta no servidor foi alterada, pode fazer com que a informação visualizada não seja a mais recente. A definição de tempos de expiração para as páginas HTML em *cache*, pela mesma ordem de ideias, impede que durante determinado tempo não haja verificação da sua alteração e consequentemente a página visualizada não seja a mais recente.

Os resultados de desempenho, após a aplicação de cabeçalhos de controlo de *cache*, têm apenas impacto no cenário da visualização repetida, como apresentado na Figura 154.

						Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 1)	36.520s	0.287s	24.718s	28631	601	36.520s	376	10,242 KB	37.019s	378	10,264 KB	$$$$$
Repeat View (Run 4)	2.542s	0.806s	2.699s	3431	337	2.542s	7	160 KB	3.499s	20	257 KB	

Figura 154: Resultados com *cache*

- **tempo de carregamento –** 2.5 segundos (menos 67.1%);
- **início da renderização –** 2.7 segundos (menos 65.8%);
- ***speed index*** – 3431 (menos 59.6%);
- **tempo de carregamento total** – 3.5 segundos (menos 68.4%);
- **bytes transferidos** – 160 Kilobytes (menos 42%).

A adição de tempos de expiração aos recursos enviados pelo servidor melhorou mais de 50% os valores das métricas no cenário da visualização, com exceção dos bytes transferidos. O número de pedidos de rede foi reduzido substancialmente, de 359 para 20. Apesar de na visualização repetida, dos cenários anteriores, existirem muitos pedidos de rede, na maior parte das vezes não era transferida a resposta por já se encontrar em *cache.*

O código implementado para alterar o cabeçalho de controlo de *cache* é o presente na Figura 155.

```
var regCache=/^(\/app\/|\/assets\/|\/bower_components\/|\/fonts\/)/;
app.use(function(req,res,next){
  if(req.method==="GET" && regCache.exec(req.url)){
    res.setHeader('Cache-Control', 'public, max-age=31557600');
  }
  next();
});
```

Figura 155: Implementação de controlo de *cache*

A função ilustrada é executada cada vez que é feito um pedido de rede ao servidor. Se o pedido for feito com o método GET e o caminho do recurso corresponder à expressão regular contida na variável "regCache", a resposta é enviada com um tempo de expiração de um ano (31557600 segundos). O início do caminho dos dados da API é "/api/", portanto as respostas não terão um tempo de expiração. A página HTML, como se verifica na Figura 156, está fora das pastas referenciadas na expressão regular, por isso também não é afetada.

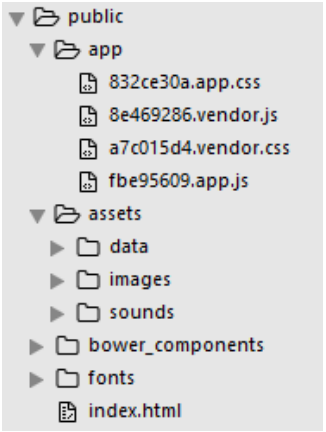


Figura 156: Estrutura de pastas da aplicação em produção

As imagens guardadas a partir da plataforma são armazenadas na *cloud* da Amazon, e para que elas tenham um tempo de expiração é preciso referir o tempo de expiração no *upload*.

```
function upload(body,path){
    s3.putObject({
        ACL: 'public-read',
        Bucket: configAWS.bucket,
        Key: path,
        Body: body,
        ContentType: getContentTypeByFile(path),
        CacheControl: 'public, max-age=31557600'
    }, function(error, response){
        if(error){
            console.log(error);
        }
        else{
            console.log('https://'+configAWS.bucket+'.s3.amazonaws.com/'+path);
        }
    });
}
```

Figura 157: *Upload* de uma imagem com tempo de expiração

O *upload* da imagem é feito com recurso a um módulo disponibilizado pela Amazon. No método que faz o *upload*, cujo código está representado na Figura 157, é definido no primeiro parâmetro, um objeto com a propriedade "CacheControl", que tem o valor do cabeçalho de controlo de *cache* com o tempo de expiração de um ano que será recebido pelo utilizador ao pedir a imagem.

8.4.5 Imagens

A otimização das imagens passa numa primeira fase pelas escolhas certas do número de cores e o formato adequado, como abordado anteriormente. Não é possível preparar a plataforma para editar esses aspetos da imagem, portanto é da responsabilidade dos utilizadores trata-la com um programa de edição de imagem. Numa segunda fase, as imagens podem ser otimizadas, ajustando as suas dimensões e utilizando de algoritmos de compressão que são processos que podem ser automatizadas nas aplicações Web.

A ferramenta utilizada para automatizar os processos de otimização de imagens foi o ImageMagick (http://www.imagemagick.org). Uma vez instalado no servidor da aplicação, instalou-se também o módulo Node gm (http://aheckmann.github.io/gm) na aplicação para comunicar com o programa de tratamento de imagem. O módulo permite comunicar com outro *software* de manipulação de imagens para além do ImageMagick, o GraphicsMagick (http://www.graphicsmagick.org). A seguir analisa-se um caso de uso onde é utilizada esta ferramenta.

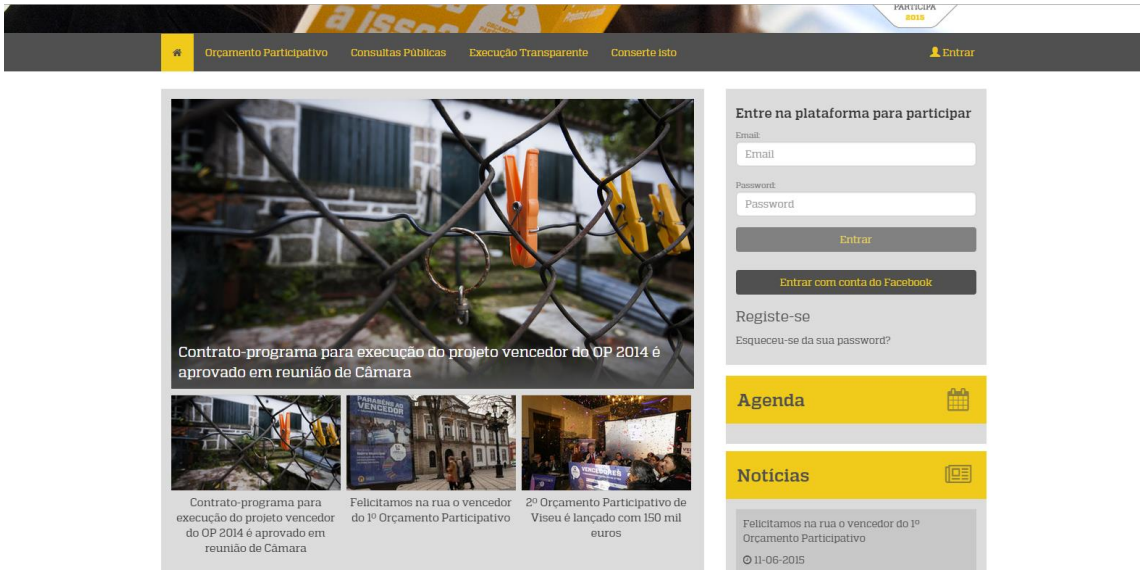


Figura 158: Caso de uso de manipulação de imagens

A maior imagem da página representada na Figura 158, tem dimensões naturais de 3600 por 2400 pixéis e um tamanho de 2.57 Megabytes. As suas dimensões na página são 722 por 40 pixéis, portanto há conteúdo transferido em excesso. A resolução do problema do excesso de bytes transferidos passou pela utilização do módulo referido para redimensionar e ajustar a qualidade das imagens, antes destas serem armazenadas na *cloud* da Amazon.

```
gm(body)
    .noProfile()
    .resize(dim[0])
    .strip()
    .interlace('Plane')
    .quality(80)
    .stream(function(err,stdout){
        handleStream(stdout,function(err,stream){
            if(err){
                return;
            }
            upload(stream,path);
        });
    });
```

Figura 159: Código da manipulação de imagens

O código da manipulação da imagem está representado na Figura 159, e realiza as seguintes mudanças:

1. A extração de informação adicional não utilizada para renderizar a imagem, como comentários e EXIF. Os métodos utilizados para remover a informação são o "noProfile" e "strip".
2. O redimensionamento para as dimensões utilizadas na sua renderização na página. Utiliza-se o método "resize" com o parâmetro da largura da imagem. A altura é ajustada automaticamente para um valor proporcional.
3. Entrelaçamento em plano para que a imagem seja carregada progressivamente, aplicado com o método "interlace('Plane')".
4. Redução da qualidade em 80% com o método "quality".

As alterações implementadas reduziram o tamanho da imagem de 2.57 Megabytes para 134 Kilobytes (menos 95%). A imagem modificada tem dimensões exageradas para a página onde é utilizada. Quanto maiores as dimensões, maiores são os ganhos. A melhoria de 95% não se aplica a todos os casos.

Os utilizadores podem não ter os conhecimentos necessários para otimizar a imagem, tendo em vista o desempenho da página, portanto é uma mais-valia que as aplicações Web automatizem as alterações apresentadas.

Para além da imagem maior, da Figura 158, existem outras três com menores dimensões. Uma das menores é igual à maior, portanto são feitos 3 pedidos HTTP em vez de 4. Antes de implementar o código de otimização, seriam transferidos 5,2 Megabytes (2.57+1.37+1.26) em imagens. Após as alterações mencionadas passaram a ser feitos 4 pedidos HTTP, pois todas as imagens transferidas têm diferentes dimensões e a serem transferidos 215 Kilobytes (menos 96%).

Apesar da aplicação Web desenvolvida ter uma página dedicada para otimizar todas as imagens de uma só vez, verificou-se que a aplicação da Liberopinion ficaria muito dependente dela, portanto foram feitas as alterações do seu código para automatizar as melhorias das imagens.

Os elementos visuais da aplicação Web ajustam-se ao tamanho da janela do *browser*, portanto é de esperar que os dispositivos móveis recebam imagens menores que os computadores e os tablets. A utilização da técnica estudada para fornecer diferentes imagens de acordo com o tamanho do ecrã, cujas *tags* estão representadas na Figura 160, tem alguns inconvenientes.

```
<picture>
  <source media="(min-width: 800px)" srcset="big.png">
  <source media="(min-width: 400px)" srcset="small.png">
  <img src="small.png" alt="Description">
</picture>
```

Figura 160: Utilização do elemento HTML "picture"

Uma página HTML com as *tags* da Figura 160, renderiza a imagem “big.png” em ecrãs com largura superior a 800 pixéis e o *browser* suporta a *tag* “picture”. Nos restantes cenários é renderizada a imagem “small.png”, uma vez que segundo o componente, ela deve ser renderizada quando a largura do dispositivo é superior a 400 pixéis e inferior a 800, quando nenhuma das *media querys* é cumprida e quando a *tag* “picture” não é suportada pelo *browser*.

A estratégia pensada para selecionar as imagens com determinadas dimensões, de acordo com as *media querys* aplicadas, foi incluir as dimensões no nome das imagens, isto é, se ela for gerada com 722 pixéis de largura e 400 de altura, o seu nome ficaria no formato “<nome da imagem>_722_400.jpg”. O componente “picture” seria o apresentado na Figura 161.

```
<picture>
    <source media="(min-width:1200px)" srcset="kxpPWNsF_722_400.jpg">
    <source media="(min-width:992px)" srcset="kxpPWNsF_588_332.jpg">
    <source media="(min-width:768px)" srcset="kxpPWNsF_692_390.jpg">
    <source media="(min-width:385px)" srcset="kxpPWNsF_309_175.jpg">
    <img src="kxpPWNsF.jpg">
</picture>
```

Figura 161: *Tag* "picture" ajustada às necessidades da aplicação

Se a imagem com a fonte (*srcset*) que cumpre condição da *media query* não existir, não é renderizado nada no componente. Este comportamento é um problema porque já há imagens na plataforma e, na altura, não foram criadas as imagens com outras dimensões, portanto no lugar delas não apareceria nada depois de ser implementada a solução. O comportamento esperado ou pretendido, seria a renderização de uma imagem por defeito nos casos em a fonte da imagem, que cumpre a *media query*, não existe.

A solução para o problema seria criar um componente HTML, cuja imagem seria carregada via javascript. Uma imagem por defeito seria carregada no caso de a imagem da condição aceite não existir. A *framework* Angular permite criar diretivas que são componentes HTML ao qual é associado código javascript. A diretiva é associada, neste caso, a uma *tag* HTML através do atributo "picture-angular", como apresentado na Figura 162.

```
<div picture-angular src-model="destaque" dimensions="[[236,133],[191,108],[226,128],[692,390],[309,175]]"></div>
```

Figura 162: *Tag* da diretiva que ajusta o tamanho das imagens

O valor do atributo "src-model" é um objeto javascript que tem a fonte da imagem. O componente é flexível, pois no valor do atributo "dimensions" definem-se as várias dimensões que a imagem deve apresentar em cinco cenários, podendo ser utilizado em qualquer página HTML da aplicação. O valor é um vetor e cada um dos seus elementos corresponde às dimensões das imagens quando ocorrem as condições mencionadas na marcação HTML da Figura 161. Por exemplo, para dispositivos com larguras de ecrã superiores a 1200 pixéis, a imagem é renderizada com 236 pixéis por 133, enquanto para ecrãs inferiores a 1200 e superiores a 992 pixéis ela terá 192 pixéis por 108. Ainda neste componente, são detetados redimensionamentos da área de visualização, cuja mudança pode fazer com que a *media query* à qual está a associada a fonte da imagem deixe de ser cumprida e seja transferida a imagem com novas dimensões. Assim o componente tem o mesmo comportamento do "picture" e nos casos em que a imagem não existe com determinadas dimensões é apresentada a imagem original. A associação do *template* e do código javascript à diretiva é feito pelo Angular.

Após implementadas as modificações descritas foram feitos novos testes à página, e obtiveram-se os resultados da Figura 163.

						Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 4)	26.489s	0.230s	24.378s	25349	611	26.489s	384	5,624 KB	26.828s	390	5,639 KB	$$$$$
Repeat View (Run 5)	9.082s	0.228s	8.197s	8566	593	9.082s	370	400 KB	9.245s	373	407 KB	

Figura 163: Resultados com otimização de imagens

Primeira Visualização:

- **tempo de carregamento –** 26.4 segundos (menos 27.2%);
- **início da renderização** – 24.3 segundos (mais 2%)
- ***speed index*** – 25349 (menos 3.7%);
- **tempo de carregamento total** – 26.8 segundos (menos 26.7%);
- **bytes transferidos** – 5,639 Kilobytes (menos 47.8%).

O tempo melhorado diz respeito à redução do tempo despendido na transferência das imagens. Analisando os gráficos em cascata dos dois testes, da Figura 164 e Figura 165, visualizam-se barras azuis maiores num, que correspondem ao tempo de transferência das imagens no teste em que não há otimização das imagens.

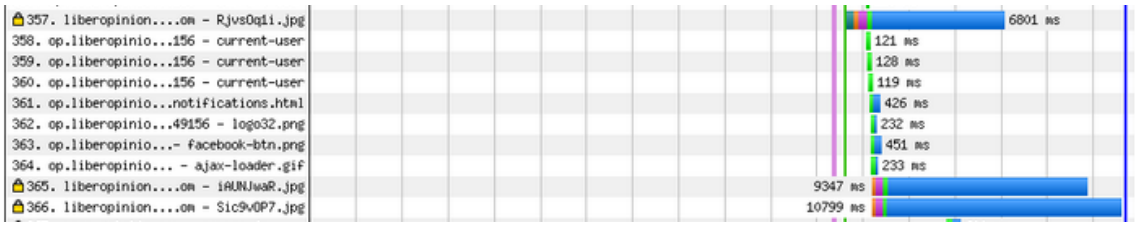


Figura 164: Tempos de carregamentos de imagens não otimizadas

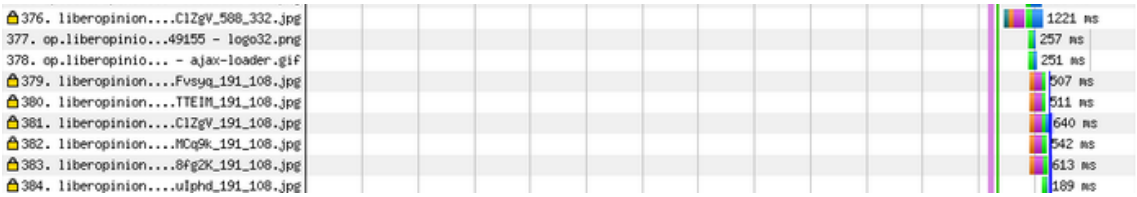


Figura 165: Tempos de carregamentos de imagens otimizadas

No gráfico do teste onde as imagens estão otimizadas, na Figura 165, verifica-se que o tempo de transferência das imagens é muito reduzido. No primeiro teste foram utilizadas três imagens, enquanto no segundo são visualizadas sete. Enquanto foi feito o desenvolvimento foi acrescentado mais conteúdo à página, o que não invalida os resultados, pois as mesmas imagens utilizadas no primeiro teste estão presentes no segundo.

Combinou-se esta implementação com as técnicas anteriores e repetiu-se o teste, obtendo os resultados presentes na Figura 166.

						Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 1)	5.973s	0.253s	3.749s	5008	286	5.973s	50	1,659 KB	8.719s	59	1,675 KB	$$$$-
Repeat View (Run 2)	2.686s	0.242s	1.649s	2100	273	2.686s	22	199 KB	3.087s	23	206 KB	

Figura 166: Gzip + combinação de recursos + otimização de imagens

Primeira Visualização:

- **tempo de carregamento –** 5.9 segundos (menos 83.7%);
- **início da renderização –** 3.7 segundos (menos 84.4%);
- ***speed index*** – 5008 (menos 80.9%);
- **tempo de carregamento total** – 8.7 segundos (menos 76.2%);

- **bytes transferidos** – 1659 Kilobytes (menos 84.6%).

Visualização repetida:

- **tempo de carregamento –** 2.6 segundos (menos 65.7%);
- **início da renderização –** 1.6 segundos (menos 79.7%);
- ***speed index*** – 2100 (menos 72.4%);
- **tempo de carregamento total** – 3 segundos (menos 67.3%);
- **bytes transferidos** – 199 Kilobytes (menos 27.8%).

Verificam-se melhorias significativas em todas as métricas dos dois cenários nos resultados apresentados na Figura 166. Pela primeira vez o tempo de carregamento na primeira visualização é inferior a sete segundos.

8.4.6 Outros

Após a implementação de técnicas de melhoria, analisou-se o gráfico em cascata, onde se identificaram dois ficheiros javascript que não estavam a ser combinados.

O ficheiro da livraria jquery, devido a incompatibilidades entre *browsers*, era carregado com uma versão diferente para o Internet Explorer 8, o que impedia que ele fosse combinado com os outros recursos. Uma vez descontinuado o suporte ao *browser*, passou a ser possível combiná-lo com os outros ficheiros javascript.

O outro ficheiro era o "socket.io.js", este tinha mais impacto no desempenho porque não estava minificado nem ofuscado. Isto acontecia porque o ficheiro estava localizado num módulo Node e pensava-se não ser possível fazer modificações sobre ele. Alguma pesquisa permitiu descobrir onde estava o ficheiro na aplicação e movê-lo para a localização de outros ficheiros javascript, para ser ofuscado e combinado com eles. Realizando outra vez os testes, obtiveram-se os resultados da Figura 167.

						Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View	5.222s	0.243s	2.981s	4130	284	5.222s	51	1,456 KB	7.983s	57	1,472 KB	$$$--
Repeat View	2.003s	0.236s	0.911s	1448	256	2.003s	20	41 KB	2.521s	22	49 KB	

Figura 167: Resultados de combinação dos ficheiros jquery e socket.io.js

Primeira visualização:

- **tempo de carregamento –** 5.2 segundos (menos 73.7%);
- **início da renderização –** 2.9 segundos (menos 88.5%);
- ***speed index*** – 4130 (menos 83%);
- **tempo de carregamento total** – 8 segundos (menos 73.5%);
- **bytes transferidos** – 1472 Kilobytes (menos 82.2%).

Visualização repetida:

- **tempo de carregamento –** 2 segundos (menos 73.7%);

- **início da renderização –** 0.9 segundos (menos 88.5%);
- ***speed index*** – 1448 (menos 83%);
- **tempo de carregamento total** – 2.5 segundos (menos 73.5%);
- **bytes transferidos** – 49 Kilobytes (menos 82.2%).

Na visualização repetida à uma melhoria significativa (82.7%) no tamanho dos recursos transferidos, porque o recurso "socket.io.js" não tinha qualquer tipo de política de controlo de *cache*. Este facto afetava também negativamente os valores de outras métricas.

A análise do gráfico em cascata é útil, pois permite identificar os ficheiros que estão a ser transferidos e o tempo que é demorado em cada um deles, possibilitando a identificação de possíveis melhorias a fazer nas aplicações Web.

8.4.7 Conclusão

Na Tabela 4 são apresentados os resultados das técnicas que melhoram os valores das métricas de performance na primeira visualização.

Técnica	Tempo de carregamento	Início da renderização	Speed Index	Tempo de carregamento total	Kilobytes Transferidos
Gzip	35.5%	-	-	-	31%
Combinação de recursos	**46.6%**	**69.8%**	**62.5%**	**47.2%**	18.2%
Imagens	26.4%	-	-	26.7%	**47.8%**

Tabela 4: Melhorias das técnicas na primeira visualização

A combinação de recursos e a ofuscação são as técnicas que mais impacto têm na melhoria da aplicação na primeira visualização. Para além de ser a técnica que mais melhora o tempo de carregamento (46.6%), também é a única que melhora as métricas que afetam a perceção humana, o início da renderização (69.8%) e o *Speed Index* (62.5%). A enorme quantidade de recursos que as grandes *single page application* têm, torna esta técnica a melhor solução para melhorar o desempenho. As técnicas de compressão Gzip e otimização de imagens melhoram também o tempo de carregamento, mas têm mais impacto na redução do tamanho dos recursos transferidos (31% e 47.8%), o que diminui o custo de acesso ao *website.*

Na Tabela 5, são apresentados os resultados das técnicas que melhoram os valores das métricas de performance na visualização repetida.

Técnica	Tempo de carregamento	Início da renderização	Speed Index	Tempo de carregamento total	Kilobytes Transferidos
Combinação de recursos	64.4%	**78.5%**	**72.5%**	68.4%	7.6%
Cache	**67.1%**	65.8%	59.6%	68.4%	**42%**

Tabela 5: Melhorias das técnicas na visualização repetida

As duas técnicas apresentadas na tabela, a combinação de recursos, ofuscação e a implementação de uma política de controlo de *cache*, têm ambas um grande impacto na

visualização repetida. As suas melhorias nos valores das métricas estão acima dos 50%, com exceção do tamanho transferido na combinação de recursos. As melhorias do tempo de carregamento são aproximadamente as mesmas (65%) nas duas técnicas. A combinação de recursos e ofuscação são as técnicas que têm mais impacto nas métricas que afetam a perceção dos utilizadores com melhorias de 78.5% no tempo de início de renderização e de 72.5% no valor do *Speed Index*. A implementação de uma boa política de *cache* diminuiu o tamanho dos recursos transferidos em 42%.

Na Tabela 6 são apresentados os resultados das melhorias na primeira visualização, combinando as técnicas.

Técnica	Tempo de carregamento	Início da renderização	Speed Index	Tempo de carregamento total	Kilobytes Transferidos
Combinação de recursos + compressão Gzip	47.6%	70.6%	62.1%	46.9%	18.2%
Combinação de recursos + compressão Gzip + otimização de imagens	83.7%	84.4%	80.9%	76.2%	84.6%
Estado final	85.6%	87.5%	84.3%	78.2%	86.4%

Tabela 6: Melhorias da combinação de técnicas

Verifica-se que na primeira combinação, que ela é a que contribui mais para a melhoria do início da renderização (70.6% dos 87.5%) e o Speed Index (62.1% dos 84.3%). A combinação com a otimização das imagens contribui para reduzir o tamanho dos recursos transferidos, que se reflete assim numa melhoria de 18.2% para 84.6%. Na Tabela 7 são apresentados os valores de desempenho, na primeira visualização, antes e depois das otimizações.

	Tempo de carregamento	Início de renderização	Speed Index	Tempo de carregamento total	Kilobytes Transferidos
Antes	36.3 segundos	23.8 segundos	26332	36.6 segundos	10812 KB
Depois	5.2 segundos	3 segundos	4130	8 segundos	1472 KB

Tabela 7: Otimização na primeira visualização

Na Tabela 8 são apresentados os valores de desempenho, na visualização repetida, antes e depois das otimizações.

	Tempo de carregamento	Início de renderização	Speed Index	Tempo de carregamento total	Kilobytes Transferidos
Antes	7.6 segundos	7.9 segundos	8500	9.5 segundos	276 KB
Depois	2 segundos	0.9 segundos	1448	2.5 segundos	49 KB

Tabela 8: Otimização da visualização repetida

O tempo de carregamento é inferior aos críticos 7 segundos, na primeira visualização. Os testes sintéticos foram executados com uma largura de banda de 5 Megabits por segundo de *download* e 1 de *upload.* Seria interessante fazer os testes sintéticos para avaliar como a aplicação se comporta em condições de rede piores, em que a largura de banda é de 780 Kilobits por segundo de *download* e 330 de *upload.*

8.5 Otimização do tempo de início da renderização

O início da renderização é essencial para indicar o progresso de carregamento da página ao utilizador. A aplicação Web otimizada, uma vez construída com a *framework* Angular, apenas inicia a renderização quando todos os ficheiros javascript são transferidos, porque é a partir deles que a página HTML é adicionada dinamicamente. O crescimento da aplicação faz com que estes ficheiros javascript cresçam e consequentemente o início da renderização seja cada vez mais tarde. A solução encontrada baseou-se em alterar a página HTML para ser renderizado o elemento presente na Figura 168, até que os ficheiros javascript fossem transferidos.

Figura 168: Loader

Na plataforma desenvolvida para auxiliar a medição da performance foi programada a execução de testes sintéticos, diariamente, aos *websites* referidos anteriormente, com as mesmas condições utilizadas nos testes sintéticos anteriores (*browser,* localização, tipo de rede e número de repetições do teste).

A implementação da solução foi feita em Setembro de 2015, o mês em que se registaram mais sessões nos *websites* covilhadecide.pt (14,806 sessões) e viseuparticipa.pt (6,896 sessões). Estes foram os *websites* escolhidos para analisar o impacto da redução do tempo de início da renderização no comportamento humano. A taxa de rejeição foi a métrica escolhida para avaliar o comportamento dos utilizadores.

Na Figura 169 são representados os valores das métricas da primeira visualização, recolhidos nos testes sintéticos executados no *website* viseuparticipa.pt, desde o dia 14 de Setembro de 2015 até 24 de Setembro de 2015.

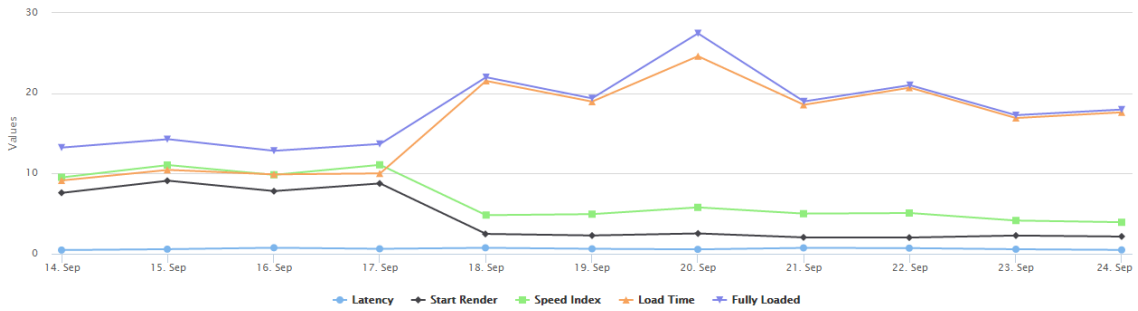


Figura 169: Diminuição do tempo de início de renderização em viseuparticipa.pt

A implementação foi realizada no dia 17 de Setembro de 2015 e no teste executado à meia-noite dia 18 de Setembro de 2015 houve uma diminuição do tempo de início da renderização de 8.76 segundos para 2.48 segundos, representado pela linha preta, do dia 17 para o dia 18. Na Figura 170, são analisadas as taxas de rejeição diárias desde o dia 3 de Setembro a 1 de Outubro, ou seja, duas semanas antes e duas depois da implementação da solução de melhoria.

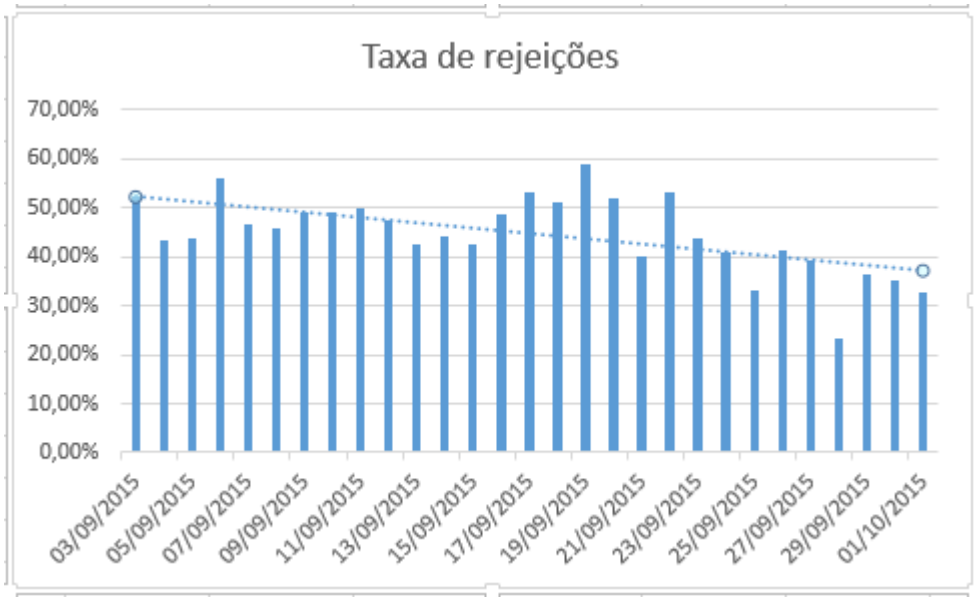


Figura 170: Impacto do tempo de início de renderização na taxa de rejeição em viseuparticipa.pt

A linha de tendência demonstra que a taxa de rejeição diminui. No primeiro dia a taxa de rejeição situa-se aproximadamente nos 50% e no último abaixo dos 40%.

Na Figura 171, são apresentados os valores das métricas da primeira visualização, recolhidos nos testes sintéticos executados no *website* covilhadecide.pt, desde o dia 5 de Setembro de 2015 até 15 de Setembro de 2015.

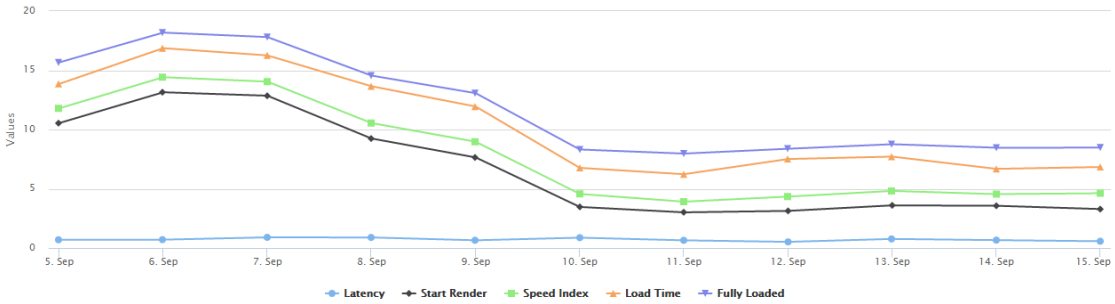


Figura 171: Diminuição do tempo de início de renderização em coviladecide.pt

A implementação foi realizada no dia 9 de Setembro de 2015 e no teste executado à meia-noite dia 10 de Setembro de 2015 houve uma diminuição do tempo de início da renderização de 7.64 segundos para 3.47 segundos, representado pela linha preta, do dia 9 para o dia 10. Na Figura 172, são analisadas as taxas de rejeição diárias desde o dia 27 de Agosto a 24 de Setembro, ou seja, duas semanas antes e duas depois da implementação da solução de melhoria.

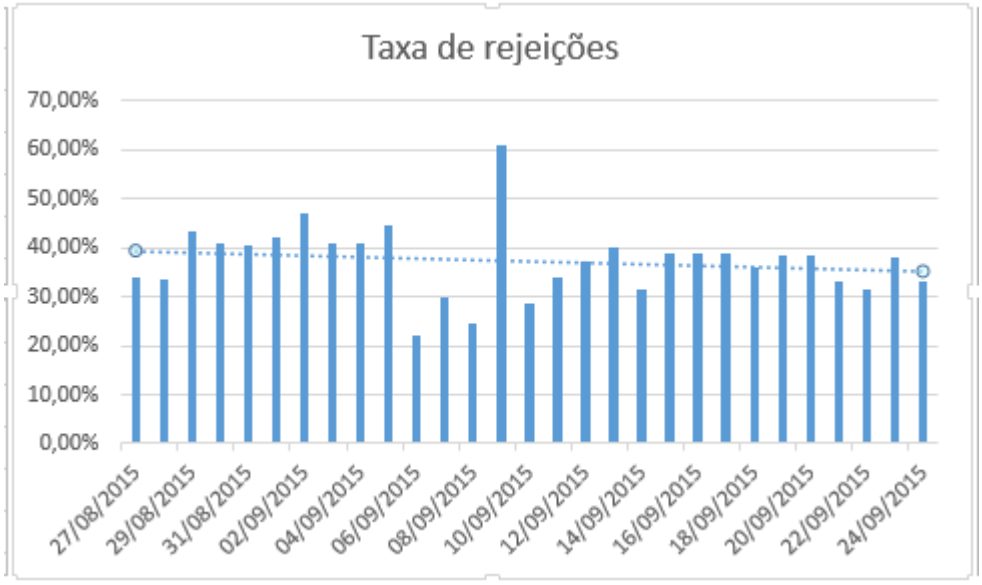


Figura 172: Impacto do tempo de início de renderização na taxa de rejeição em covilhadecide.pt

A linha de tendência demonstra que a taxa de rejeição diminui. No primeiro dia a taxa de rejeição situa-se aproximadamente nos 40% e no último perto dos 35%.

Os dados analisados comprovam que o início da renderização tem impacto no comportamento humano, quanto mais tarde se iniciar maior é a probabilidade de os utilizadores desistirem de aceder ao *website*. A combinação da análise dos resultados dos testes sintéticos com os dados recolhidos de utilizadores reais possibilitam tirar conclusões. Apesar dos valores das métricas de performance apresentados nos testes sintéticos não serem os valores que a maior parte das sessões terá, quanto melhores forem esses valores, melhores serão os valores nas sessões dos utilizadores.

Factos analisados:

- Viseuparticipa.pt;
    - O início da renderização é reduzido de 8.7 para 2.4 segundos;
    - A taxa de rejeição diminui de 50% para 40%, segundo a linha de tendência;
    - A renderização completa da janela de visualização dá-se aos 5 segundos.
- Covilhadecide.pt;
    - O início da renderização é reduzido de 7.6 para 3.4 segundos;
    - A taxa de rejeição diminui de 40% para 35%, segundo a linha de tendência;
    - A renderização completa da janela de visualização dá-se aos 5 segundos.

A solução implementada tem maior impacto no *website* viseuparticipa.pt, havendo uma redução mais abrupta da taxa de rejeição. A taxa de rejeição em covilhadecide.pt era menor e manteve-se abaixo da taxa de viseuparticipa.pt. Quanto mais cedo se iniciar a renderização, menor é a taxa de rejeição.

8.6 Otimização de pedidos assíncronos

Após as melhorias implementadas na secção anterior, com as quais se obtiveram reduções significativas dos valores das métricas de desempenho no global, identificaram-se alguns problemas com os pedidos assíncronos de informação num *website* com a plataforma Liberopinion. À medida que a aplicação Web testada vai sendo utilizada e é acrescentado conteúdo, o tamanho dos recursos a serem transferidos assincronamente aumenta, o que provoca maior processamento por parte do servidor. A seguir descrevem-se alguns problemas associados a pedidos assíncronos que requerem elevado tempo de processamento, soluções e os resultados da implementação dessas soluções. A abordagem seguida para avaliar o desempenho, antes e depois das soluções tomadas, consistiu na replicação do ambiente da aplicação, onde se identificaram os problemas de desempenho, noutra máquina. Isto é, foi feita a cópia dos seus dados para um novo sistema de gestão de base dados, ao qual acedia a aplicação onde se avalia o desempenho das suas páginas enquanto se implementam as soluções. A avaliação do desempenho é feita com recurso à ferramenta do *browser* Google Chrome, com uma ligação ADSL de 30 Megabits por segundo.

8.6.1 Leitura de votos por tipo de submissão

No capítulo da demonstração da aplicação teste, apresentou-se uma fase do orçamento participativo que consistia na votação nos projetos. A votação pode ser feita através de SMS, da página Web da plataforma e presencialmente numa assembleia participativa. Parte da página do orçamento participativo e da listagem dos projetos apresenta a um administrador no *backoffice*, a fonte dos votos do orçamento participativo.

Figura 173: Demonstração da fonte dos votos

Na Figura 173, apresenta-se um total de 9423 votos, em que 4266 deles são feitos em assembleia participativa e 5157 a partir da aplicação Web por utilizadores registados. O tempo demorado desde que é solicitado o pedido até ser recebida a resposta é em média 27 segundos. A Figura 174 apresenta gráfico em cascata, representado na Figura 173, dos pedidos de rede solicitados quando se acede à página com o componente de amostragem de votos por fonte.

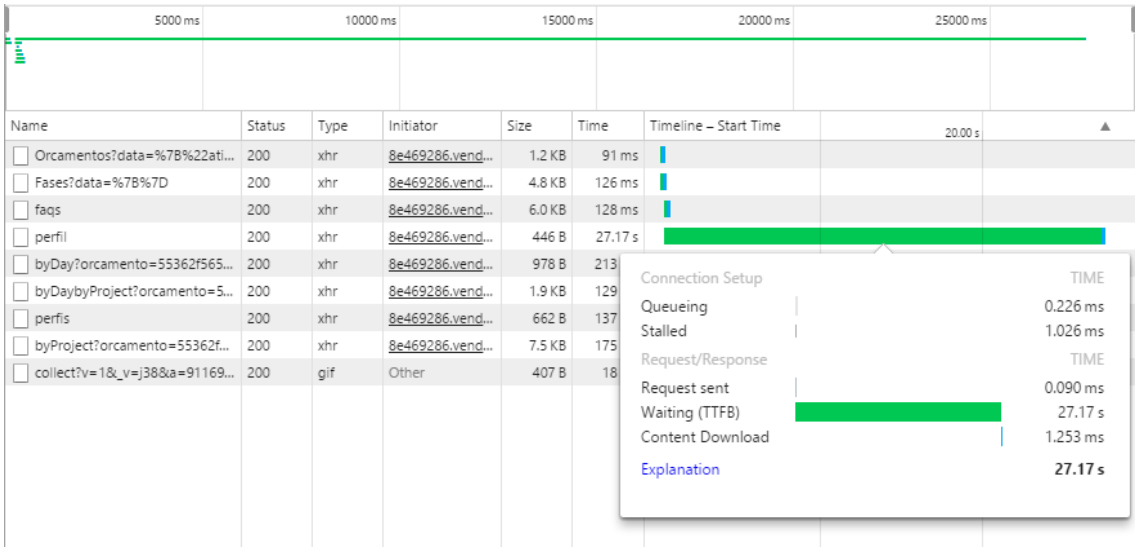


Figura 174: Pedido de votos por tipo de fonte

Repare-se que a resposta do pedido demora cerca de 27.17 segundos a ser obtida, e é uma das que tem menor tamanho (446 Bytes) em comparação com a maior que tem 7.5 Kilobytes e é transferida em apenas 175 milissegundos. A resposta do pedido de votos com fonte descriminada demora aproximadamente 1.2 milissegundos a ser transferida, mas o tempo de processamento de resposta, ilustrado na barra do TTFB (*time to first byte*), é de 27.17 segundos. As melhorias de performance no *frontent* estudadas dificilmente teriam melhorias significativas neste pedido, porque o tamanho do conteúdo já é reduzido e a resposta sendo dinâmica não pode ser armazenada em *cache* por longos períodos de tempo. A solução está na análise do código do servidor responsável por quase todo o tempo de espera.

Analisando o código do lado do servidor, conclui-se que o elevado tempo do processamento da resposta deve-se à forma como está estruturada a informação na base de dados, o que complica a descriminação dos votos pela sua fonte.

O sistema de gestão de base de dados utilizado na aplicação é o Mongo, uma das base dados não relacional, classificada com o termo NoSQL. Interessa referir que não é possível fazer uma pesquisa a mais do que uma coleção (o termo que corresponde à tabela nas bases dados relacional) num acesso e não há garantias de haver consistência de dados entre duas delas. Ou seja, a consistência dos dados deve ser garantida através do código da aplicação Web.

As coleções em causa, no problema identificado, são os votos, os orçamentos participativos e os utilizadores. Os votos contêm o identificador do utilizador ou número de telemóvel que fez a votação, o identificador do objeto de votação, o identificador do orçamento participativo e a data de submissão. No processamento da resposta é preciso obter o identificador do orçamento participativo que é utilizado para extrair os votos efetuados. O número de operações aumenta ao descriminar a fonte dos votos. Os votos feitos por SMS, são distinguidos pelo campo do número de telemóvel, mas não é possível distinguir os votos feitos por utilizadores com perfil assembleia participativa. Portanto é necessário procurar na informação dos utilizadores que fizeram votos, os que são assembleia participativa, para incrementar na resposta o número de votos desse tipo de fonte.

Inicialmente não havia problemas de desempenho com o processamento desta resposta, uma vez que havia poucos utilizadores e consequentemente poucos votos. Quando identificado o problema, na plataforma em causa já tinham sido submetidos 9422 votos e 5154 utilizadores tinham-se registado, o que provocou o aumento do processamento e consequentemente o tempo de tratamento da resposta. A resposta como estava a ser tratada, garantia que o número de votos estava sempre correto porque eram contabilizados e verificada a sua fonte.

A solução passou por incluir o campo booleano “assembleia” no voto para identificar os votos que tinham como fonte a assembleia participativa. Embora reduzido o processamento a ser feito no servidor, entendeu-se que a pesquisa efetuada à base de dados podia ser simplificada. Incluíram-se campos no orçamento participativo e nos projetos com os valores do número total de votos, os votos na plataforma, por SMS e em assembleia participativa. O valor destes campos é alterado quando é submetido um voto ou anulado, garantindo a consistência dos votos. A inserção de um voto e a atualização da informação do orçamento participativo são duas operações na base dados. Nada garante que não ocorre uma falha de *hardware* e uma delas não é executada. Para ultrapassar este problema criou-se uma função que é executada hora a hora, para contabilizar o número de votos e compara-os com os valores dos campos orçamentos participativos e projetos, corrigindo-os no caso de não coincidirem. Implementada a solução, o pedido de rede efetuado para obter os votos das várias fontes deixa de ser necessário, uma vez que essa informação passa a estar na resposta do pedido que obtém os detalhes do orçamento participativo, efetuado também nessa mesma página.

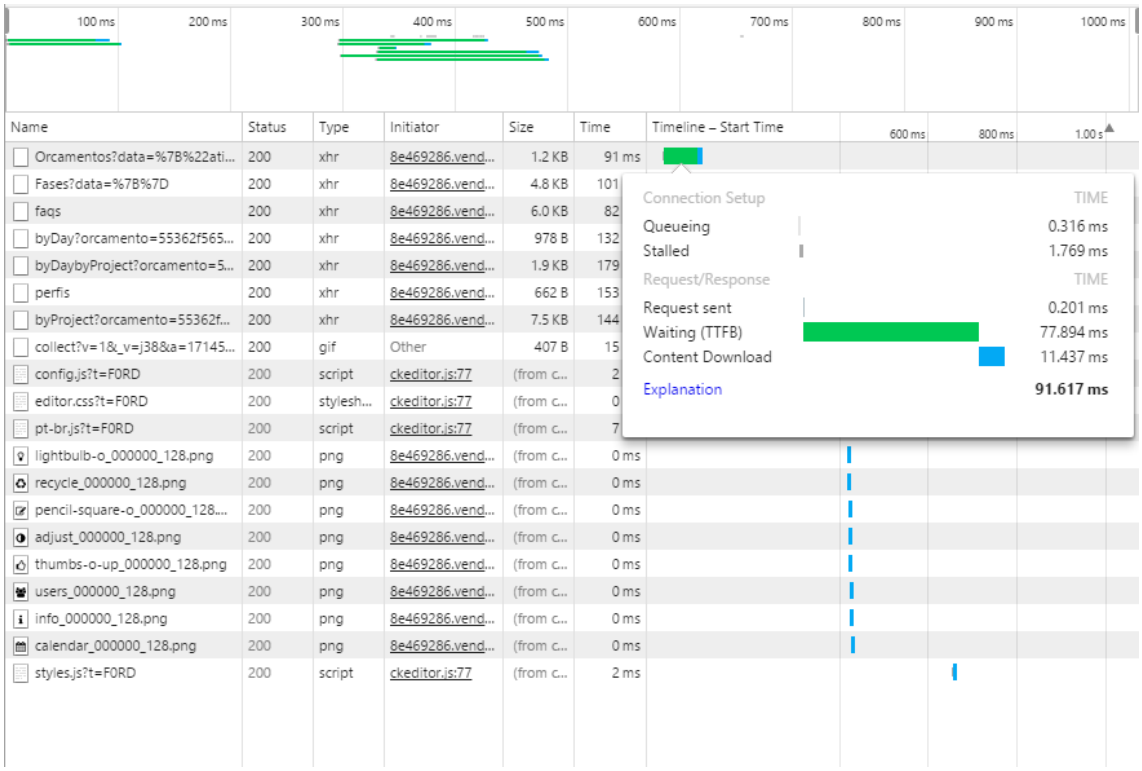


Figura 175: Sem pedido de votos descriminados por fonte

A solução implementada acrescenta apenas mais uns bytes na resposta do pedido que tem os detalhes do orçamento participativo. O tempo demorado a carregar toda a informação assíncrona é menos de meio segundo, como se verifica na Figura 175, e a informação dos votos é apresentada em menos de 100 milissegundos no acesso à págin. Comparado com os aproximadamente 27 segundos no primeiro cenário, houve uma melhoria de cerca de 98%, no tempo de carregamento da informação assíncrona da página.

O tempo de carregamento em causa diz respeito à navegação para a página a partir de outra da mesma aplicação. Tratando-se de um *single page application*, entrar na aplicação através desta página obrigava a que tivessem de ser transferidos recursos javascript, CSS e HTML, aumentando o tempo de carregamento dela.

8.6.2 Listagem de utilizador

Uma vez mais no *backoffice*, a informação presente na tabela dos utilizadores, é carregada toda de uma só vez. Ao constatar a demora do carregamento numa instância da plataforma, onde já existiam 5154 utilizadores registados, decidiu-se fazer a medição das características do pedido que obtêm tal informação.

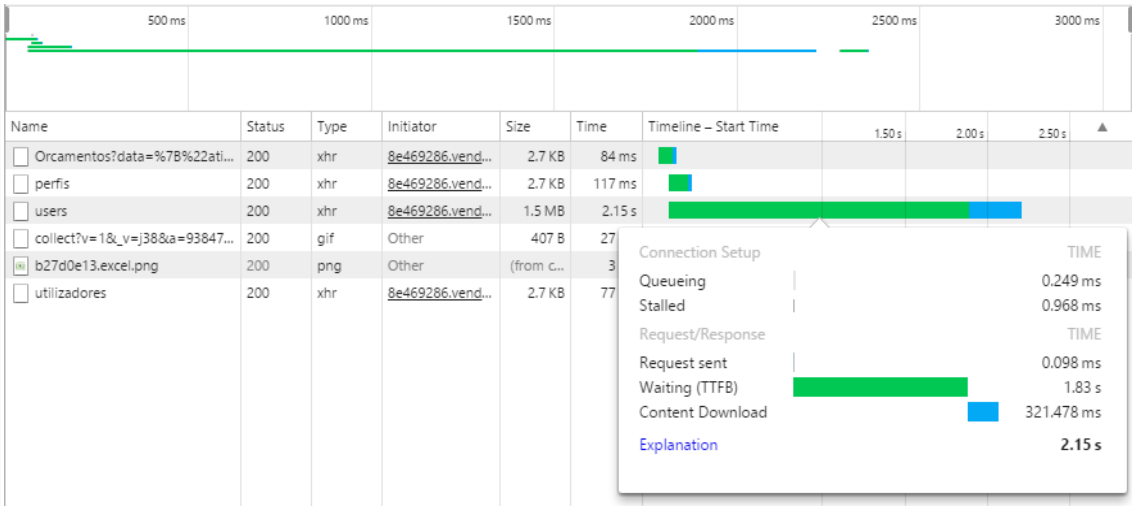


Figura 176: Carregamento completo de utilizadores

Na análise dos vários pedidos assíncronos, no gráfico em cascata da Figura 176, na navegação para a página dos utilizadores, identifica-se o pedido que obtém os utilizadores como o mais demorado com 2.15 de tempo de carregamento, sendo transferidos 1.5 Megabytes. O servidor comprime com Gzip as respostas, portanto não há forma de reduzir o tamanho, enviando a mesma informação. Embora os 2.15 segundos seja um tempo de espera aceitável, ele quebra a experiência de utilização e a tendência é ele crescer conforme se vão registando mais utilizadores na plataforma.

Surgem assim duas soluções para resolver o problema. Uma vez que a listagem de utilizadores é paginada e apenas são apresentados 10 registos de cada vez, apenas devem ser carregadas esses registos no acesso à página. As ações que levem a alterar os registos apresentados na tabela, como mudança de página ou ordenação, implicam que tenha de ser realizado um novo pedido ao servidor, estando o utilizador sempre sujeito aos tempos de latência e transferência de conteúdo. A tabela de utilizadores está representada na Figura 177.

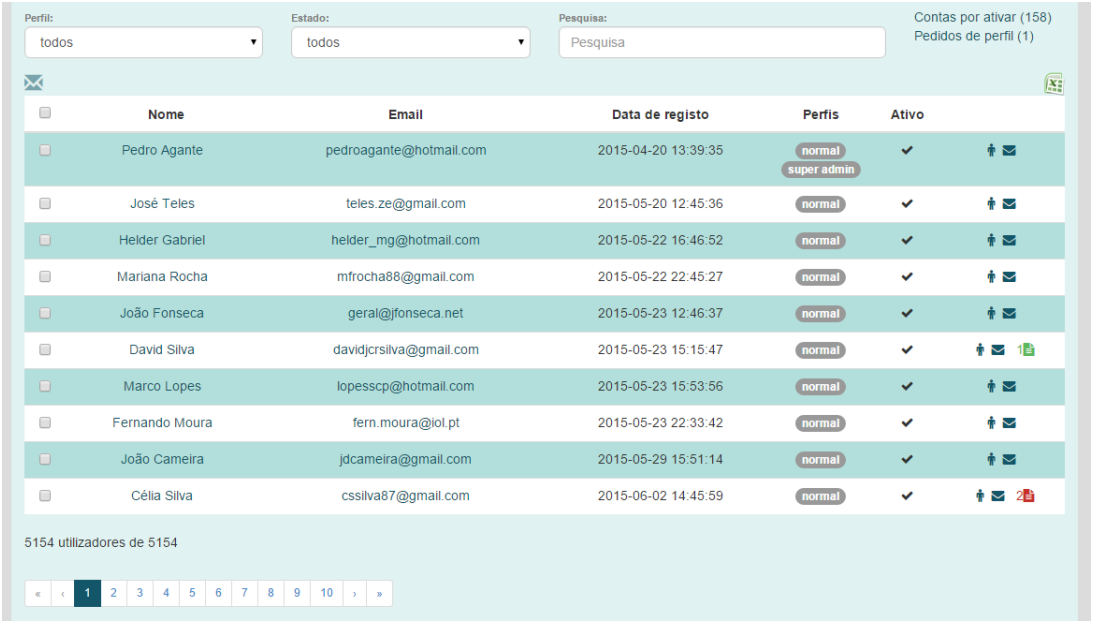


Figura 177: Listagem de utilizadores

O conteúdo visualizado pode ser alterado escolhendo outra página, um filtro ou clicando nos *links* do canto superior direito da Figura 177. Os *links* contêm o número de contas ativas e número de pedidos de perfil que são somados no *browser*, após receber a informação de todos os utilizadores.

A atualização da tabela de utilizadores provoca o processamento do lado do cliente, existindo manipulações em javascript e DOM para alterar os registos presentes na tabela. A solução sugerida obriga a que todo o código que faz este processamento deixe de ser utilizado, o que tem bastante impacto.

Uma outra solução é fazer um primeiro pedido com informação de apenas 10 utilizadores, que são apresentados na tabela mal é recebida a resposta, e depois fazer um segundo pedido com a informação dos restantes utilizadores. O segundo pedido feito não tem melhorias a nível de tempo de carregamento, o que é transparente para o utilizador. Nesta solução a informação na tabela é visualizado mais cedo, indicando ao utilizador que a página está pronta para ser utilizada. Existe conteúdo que é transferido escusadamente e aumenta o tempo de espera, pois o utilizador, em princípio, não vai percorrer todas as páginas da listagem de utilizadores. Outro problema é o facto de no acesso à aplicação através de redes móveis 3G, haver custos por cada Megabyte transferido. A solução pode não ser amigável para utilizadores que acedem à página através de redes móveis, mas o tipo de utilizador do *backoffice* é um funcionário público que acede à aplicação, a partir de um escritório onde tem acesso a um computador, no seu horário de trabalho, com outras condições de rede. Devido às características de acesso do habitual administrador da plataforma e ao menor impacto na implementação desta solução, esta foi a implementada

No decorrer da implementação identificou-se um problema, se o utilizador tentar selecionar um dos campos para filtrar ou selecionar outra página, antes que o pedido em *background* seja tratado, terá que esperar que ele seja tratado. Face ao tempo que é demorado a carregar, nos testes efetuados (2.15 segundos), ocorrerá com pouca frequência, mas mesmo assim é apresentado um indicador de progresso, um *loader*, avisando o utilizador que a informação está a ser carregada. O gráfico em cascata da página, com a solução implementada tem o aspeto representado na Figura 178.

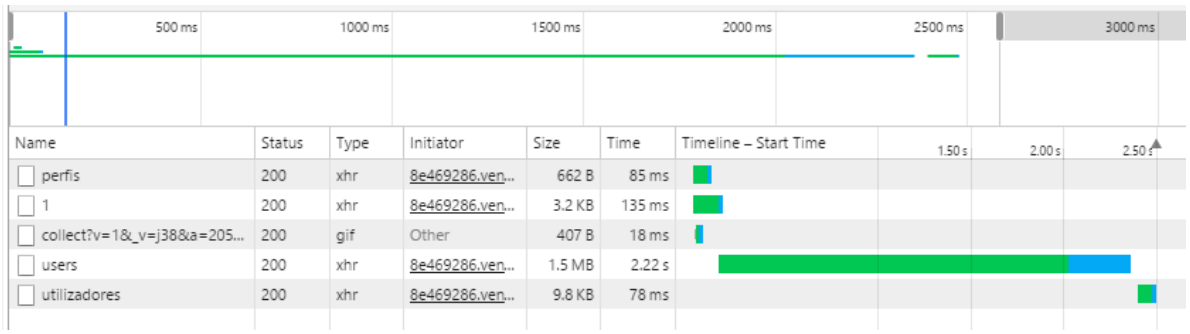


Figura 178: Carregamento da página de utilizadores após solução implementada

O pedido com o nome "1", diz respeito ao carregamento inicial da primeira página da tabela de utilizadores. Na resposta deste pedido estão também o número total de utilizadores, de contas por ativar e de pedidos de perfil pendentes. Esta informação era processada do lado do *browser*, com base na informação dos utilizadores recebida, portanto era necessário esperar

o carregamento do pedido em *background*. Por causa do tempo de espera da resposta com a informação dos utilizadores, decidiu-se fazer esse processamento do lado do servidor. Desta forma é evitado também o processamento e gasto de recursos do lado do cliente. O utilizador tem a perceção que a página está carregada em menos de 200 milissegundos, nas condições de rede utilizadas, mas em contrapartida o tempo de carregamento mantem-se praticamente igual.

8.7 Monitorização

As alterações na plataforma Liberopinion são constantes e os administradores podem adicionar dinamicamente conteúdo às páginas Web. Estes factos podem fazer variar os valores das métricas de performance, portanto é essencial monitoriza-los.

A aplicação Web que recolhe os dados de desempenho dos vários *websites* que utilizam a plataforma Liberopinion, tem uma secção onde são comparados os valores das métricas. Os *websites*, utilizando a mesma plataforma, são transferidos os mesmos recursos (javascript, CSS e HTML) e o conteúdo dinâmico adicionado pelos administradores. A secção da aplicação, apresentada na Figura 179, permite analisar e comparar o impacto do conteúdo dinâmico adicionado nos vários *websites* no valor das métricas e ajuda a planear quais devem ser otimizados com mais urgência.

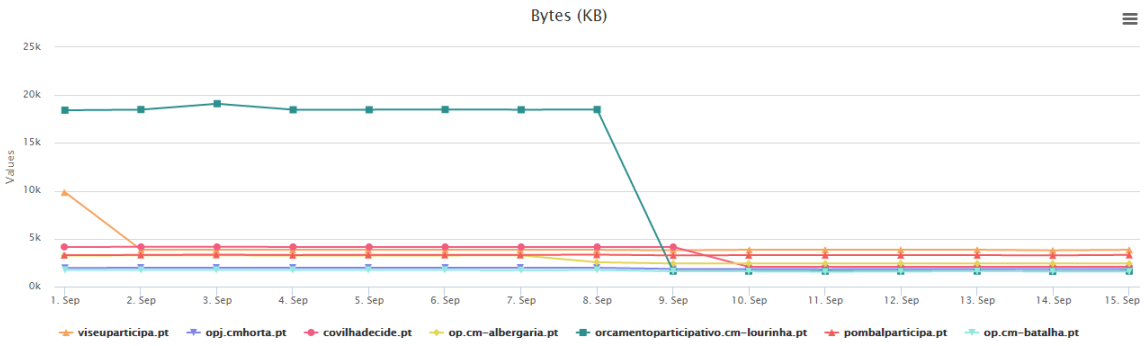


Figura 179: Comparação do conteúdo transferido em vários *websites*

Por exemplo, na Figura 179, é comparado o tamanho dos conteúdos transferidos no primeiro acesso à página inicial dos vários *websites* analisados. No *website* orcamentoparticipativo.cm-lourinha.pt eram transferidos 20.000 Kilobytes de conteúdo, até o dia 8 de Setembro. Identificou-se que o problema era o tamanho das imagens na área dos destaques. Apesar de a plataforma Liberopinion ter sido preparada para otimizar as imagens inseridas nela, antes de fazer a atualização no *website* já existiam imagens não otimizadas. A solução para diminuir o valor da métrica em causa passou pela reinserção das mesmas imagens dos destaques para que a plataforma pudesse fazer os devidos tratamentos de imagem. No dia 9 de Setembro de 2015 nota-se que há um declínio abrupto do número de bytes transferidos, aproximando-se dos valores da mesma métrica nos outros *websites*, abaixo dos 5,000 Kilobytes.

Na Figura 180, está representado o impacto da melhoria efetuada, são apresentados os dados de utilizadores reais sobre o número de acessos e o tempo de carregamento de página por dia, em forma de gráfico, entre o dia 28 de Agosto e 27 de Setembro, no *website* do Orçamento Participativo da Lourinhã (http://orcamentoparticipativo.cm-lourinha.pt).

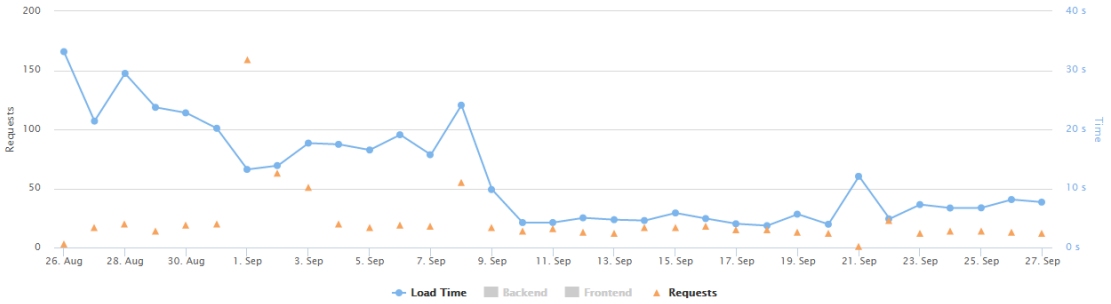


Figura 180: Dados de performance do *website* de Lourinhã

Durante o período demonstrado na Figura 180, verifica-se que a partir do dia 8 de Setembro, o tempo médio de carregamento diminui. Antes desse dia o tempo de carregamento variava entre os 15 segundos e os 31 segundos, mas a partir daí começou a variar entre 5 a 12 segundos.

9 Conclusão

9.1 Síntese e Conclusões

A performance é um requisito na construção de um *website*. Os tempos de espera afetam a perceção humana, que em demasia pioram a experiência de navegação na Web. Uma boa experiência de utilização é essencial para provocar o bem-estar nos utilizadores. O desempenho de uma aplicação afeta o número de páginas visualizadas por sessão, o número de utilizadores repetidos, a posição do *website* em motores de busca e o valor das vendas no caso das lojas online.

A medição pressupõe o conhecimento dos vários processos envolvidos na navegação Web e das métricas a avaliar. Os processos como a pesquisa DNS, o estabelecimento de conexão TCP e a transferência de conteúdo contribuem para o aumento do tempo de resposta de uma página Web. A medição das métricas relacionadas com os processos envolvidos no acesso a uma página ajuda a identificar problemas de desempenho. Entre as métricas a avaliar estão o tempo de início de renderização, o Speed Index, o tempo de carregamento, o tempo de carregamento total e o tamanho do conteúdo transferido. O tempo de carregamento indica o momento em que o *browser* considera que as páginas Web estão prontas a ser utilizadas e coincide com o momento em que são transferidos os recursos presentes na página HTML. Este tempo não espelha a verdadeira experiência de utilização, os momentos em que se inicia a renderização e o momento em que os elementos da janela de visualização estão todos desenhados têm mais impacto na perceção do utilizador do que o tempo de carregamento.

A monitorização do desempenho pode ser feita, utilizando testes sintéticos, que são programas que extraem os valores das métricas de performance, e recolhendo de dados de

performance de utilizadores reais. Os testes sintéticos são executados, normalmente, nas mesmas condições (localização geográfica, *browser*, dispositivo, ligação de rede, etc), o que os torna a solução ideal para estabelecer limites para os valores de performance e avaliar os *websites*, antes de serem colocados em produção. A monitorização de utilizadores reais (RUM), permite identificar o impacto do desempenho no seu comportamento. Com eles, também se identificam problemas de acesso dos utilizadores que podem ter ou não a ver com as características de acesso, como por exemplo, a localização geográfica.

As ferramentas do *browser* de apoio ao desenvolvimento, o Webpagetest e o Google Analytics foram soluções encontradas para analisar a performance dos *websites*. As ferramentas do *browser* auxiliam o programador a identificar os pontos críticos de degradação do desempenho enquanto se desenvolve uma aplicação Web. O Webpagetest é uma plataforma online onde são feitos pedidos para execução de testes sintéticos a qualquer página Web, em diferentes condições de acesso. O Google Analytics recolhe informação sobre o desempenho e a utilização de um *website*, portanto, é a plataforma ideal para analisar o comportamento dos utilizadores face aos tempos de espera de acesso às páginas Web. Na tentativa de auxiliar a análise de dados dos dois métodos de recolha estudados, criou-se uma plataforma Web. O Webpagetest tem a lacuna de não permitir analisar a evolução dos valores das métricas ao longo do tempo. A plataforma desenvolvida integra uma API sua, que permite pedir a execução de testes sintéticos e os seus resultados, ultrapassando a lacuna identificada no Webpagetest.

As técnicas de melhoria de desempenho no *frontend* resolvem grande parte dos problemas de performance e baseiam-se muito resumidamente em reduzir o tamanho do conteúdo transferido, reduzir o número de pedidos de rede e evitar o bloqueio da transferência de recursos. A redução do tamanho dos recursos é garantida com a utilização da compressão GZIP, a minificação, a ofuscação e otimização de imagens. Quantos mais pedidos de rede forem feitos, mais impacto tem a latência da rede no carregamento das páginas Web. A combinação dos recursos da mesma natureza (javascript, CSS e imagens) e a utilização de uma política eficiente de controlo de *cache* são soluções para reduzir o número de pedidos de rede. A globalização de um *website* torna maior a distância física entre o servidor e os clientes mais impactante na sua performance, o que pode ser atenuado com a utilização de um CDN. O pedido de recursos a vários domínios pode ser benéfico porque aumenta o número de pedidos de rede paralelos, feitos pelo *browser*, mas obriga também a que sejam feitos mais pedidos ao DNS, portanto é importante escolher acertadamente o número de domínios que fornecem os recursos das páginas Web. Uma estruturação da página HTML tendo em conta o comportamento bloqueador do *browser* na transferência de recursos javascript e CSS pode também ser uma mais-valia.

A recolha de dados de desempenho dos utilizadores que acederam aos *websites* que utilizam a plataforma Liberopinion permitiram tirar uma conclusão que já havia sido provada em outros estudos estatísticos apresentados [9] [10]. O tempo de carregamento tem impacto na taxa de rejeição. Quanto mais tempo demorar a carregar a página, maior é a probabilidade de o utilizador sair do *website* depois de visualizar a primeira página. A taxa de rejeição aumenta à medida que os tempos de carregamento aumentam e estabiliza para tempos de carregamento superiores a 7 segundos, mantendo-se a uma taxa de rejeição de 30%. Isto revela que todas as melhorias que diminuam o tempo de carregamento para tempos inferiores a 7 segundos terão um impacto positivo no comportamento dos utilizadores.

A otimização da plataforma Liberopinion consistiu na utilização das técnicas de melhoria de desempenho apresentadas. A plataforma Liberopinion é uma *single page application*, pelo que a sua lógica situa-se quase toda do lado do cliente, obrigando a haver uma transferência de todo o seu conteúdo no primeiro acesso. Quer isto dizer que as otimizações têm impacto em todas as páginas da aplicação. A avaliação consistiu em analisar os resultados dos testes sintéticos, onde as condições de acesso são menos variáveis, à página inicial de um *website* com a plataforma Liberopinion. São analisados os cenários da primeira visualização e da visualização repetida, quando já há recursos armazenados em *cache*.

A utilização da compressão GZIP teve melhorias significativas na primeira visualização, onde há mais conteúdo a ser transferido, reduzindo 31% do tamanho dos recursos transferidos e 35% do tempo de carregamento. Esta técnica foi a q menos complexa de implementar, a correta configuração do servidor permitiu que as suas respostas passassem a ser comprimidas com GZIP antes de serem enviadas.

A técnica de otimização que mais impacto teve na melhoria da aplicação foi a combinação de recursos, tanto na primeira visualização como na visualização com recursos em *cache*. Uma das características das *single page application* é a grande quantidade de recursos a serem transferidos e a redução do número de pedidos rede é crucial para evitar os tempos de latência da rede. Na primeira visualização, a redução de 371 pedidos de rede para 56, otimizou 46.6% do tempo de carregamento, 69.8% do tempo de início de renderização, 62.5% do Speed Index e 47.2% do tempo de carregamento total. Também reduziu o tamanho dos recursos transferidos em 18.2% porque após a combinação de recursos foi realizada a ofuscação no ficheiro resultante. Na visualização repetida, a técnica otimizou 64.4% do tempo de carregamento, 78.5% do início da renderização, 72.5% do Speed Index e 68.4% do tempo de carregamento total. Esta técnica é essencial para o bom desempenho de aplicações com vários recursos javascript e CSS. A combinação de recursos envolveu uma pequena aprendizagem do funcionamento do programa de automatização, mas o *software* existente facilitou a sua implementação.

A implementação de uma boa política de controlo de *cache*, a par com a combinação de recursos, resultou em melhorias significativas na visualização repetida. A sua implementação deve ser cuidada para que os utilizadores, no acesso à página, não vejam informação desatualizada. A sua utilização evitou a execução de pedidos de rede na visualização repetida, o que otimizou 67.1% do tempo de carregamento, 65.8% do tempo de início de renderização, 59.6% do Speed Index, 68.4% do tempo de carregamento total e 42% do tamanho da aplicação. O grau de complexidade da sua implementação depende da estruturação da aplicação, a utilização da *framework* expressjs permitiu configurar os cabeçalhos relativos à política de *cache* nas respostas dos pedidos, cujo URL respeita uma simples expressão regular.

A otimização das imagens de um *website* é a técnica que mais impacto teve no tamanho dos recursos transferidos. A sua implementação diminuiu 47.8% do tamanho dos recursos transferidos na primeira visualização, que consequentemente reduziu 26.5% do tempo de carregamento e do tempo de carregamento total. A otimização das imagens reduz o tempo da sua transferência e uma vez que na visualização repetida elas já se encontram em *cache*, não é despendido nenhum tempo na transferência nesse cenário. A transferência de cada Megabyte tem um custo para o utilizador, portanto é de todo importante reduzir o tamanho dos recursos

transferidos. Por vezes a redução do tamanho não tem impacto nas restantes métricas temporais e não degrada a experiência de utilização, mas deve-se dar-lhe importância para que não hajam grandes custos para os utilizadores, o que pode provocar o seu descontentamento. Foi a técnica de otimização que revelou uma maior complexidade de implementação. A máquina onde está hospedada o *website* precisa de ter *software* específico e devem ser identificados os tamanhos de todas as imagens para diferentes larguras de ecrãs, uma vez que o *website* é *responsive*. Cada vez que é feito um *upload* de uma imagem no *backoffice* é necessário indicar todas as dimensões possíveis em que ela vai ser visualizada no *website*, para que seja transferido o mínimo número de bytes em todos os cenários. No *frontend* foi necessário fazer alterações para que fosse escolhida programaticamente a imagem com as dimensões corretas em cada uma das situações.

A implementação das técnicas de melhoria do frontend permitiram melhorar as métricas desejadas (tempo de carregamento, início de renderização, speed índex, tempo de carregamento total e tamanho) em mais de 70%, na primeira visualização e na visualização repetida, onde o acesso é feito com recursos em *cache*. No entanto, na análise da aplicação Web melhorada, identificaram-se problemas de desempenho, relacionados com processos ocorridos no *backend*, que não seriam resolúveis utilizando técnicas de melhoria de performance. Apesar das técnicas de melhoria do *frontend* serem eficientes, nada pode ser feito nos casos em que há processamento intensivo no servidor. As técnicas de otimização de base de dados e a estruturação correta da API de dados são também essenciais para garantir o bom desempenho das aplicações Web.

Na monitorização foi importante a utilização das duas técnicas de recolha de dados. Com os testes sintéticos identificaram-se alterações dos valores das métricas e na análise dos dados recolhidos dos utilizadores reais avaliou-se o seu impacto na utilização do *website.* Na otimização do tempo inicial de renderização identificou-se o seu impacto na taxa de rejeição. Quanto mais cedo se inicia a renderização menor é a taxa de rejeição. A diminuição do início da renderização de 8.7 para 2.5 segundos reduziu de 50% para 40% a taxa de rejeição no viseuparticipa.pt. Em covilhadecide.pt houve uma diminuição menos acentuada da taxa de rejeição, de 40% para 35%, quando o início da renderização passou a ser aos 3.5 segundos em vez dos 7.64 segundos antes da otimização.

Revendo os objetivos traçados inicialmente, eles foram todos cumpridos. No entanto, a otimização de desempenho é um processo contínuo e no decorrer do trabalho desenvolvido identificaram-se melhorias e tarefas a realizar futuramente.

9.2 Trabalho futuro

O desempenho de uma aplicação Web é um aspeto que deve ser monitorizado. Os administradores acrescentam constantemente conteúdo ao *website*, portanto é importante verificar se os valores das métricas de performance não aumentam descontroladamente. A plataforma Web melhorada, estando em constante evolução, é necessário executar testes sintéticos para antever algum problema de desempenho, antes das alterações serem colocadas em produção. A utilização da plataforma desenvolvida e o Google Analytics é essencial para manter a mesma experiência de utilização num *website*.

Há várias funcionalidades e melhorias identificadas, que podem ser implementadas na plataforma desenvolvida, para melhorar a análise de desempenho.

- Há necessidade de utilizar outras plataformas para completar a análise de desempenho para além da plataforma de medição desenvolvida. A plataforma não está preparada para avaliar o comportamento dos utilizadores e a execução dos testes sintéticos é realizada com recurso à API do Webpagetest. Uma solução para tornar a plataforma independente do Webpagetest, é utilizar o projeto disponibilizado em https://github.com/WPO-Foundation/webpagetest numa máquina dedicada. Outra solução é resolver os problemas associados com as condições de execução do script Phantomjs criado ao longo do trabalho deste projeto.
- A exportação dos dados visualizados nos gráficos e nas tabelas para um formato CSV, aumentaria a portabilidade dos dados, facilitando a sua utilização em programas de análise de estatística.
- Por vezes ocorrem erros, e uma página pode demorar muito mais tempo a carregar que o habitual, o que complica a análise da informação de desempenho. Por exemplo, se há 4 visualizações de página que demoraram 2 segundos e 1 visualização em que o tempo de carregamento foi 70 segundos, a média será de 15.6 segundos. Seria interessante que a plataforma estivesse preparada para filtrar os dados por percentis ou utilizar outras técnicas da estatística para analisar os dados.
- Há interesse em procurar padrões de tempos de carregamento, ou seja, identificar problemas reincidentes em determinadas condições de acesso, como por exemplo, uma determinada região geográfica.
- A recolha de dados de performance sobre todos os recursos transferidos, obrigaria a que os dados a enviar, para a plataforma, aumentassem exponencialmente, degradando o desempenho dos *websites* analisados. Ainda assim seria importante recolher essa informação, para identificar os recursos que afetam mais o tempo de carregamento das páginas, nos *browsers* dos utilizadores. É necessário analisar o impacto de recolher estes dados face aos benefícios de os recolher.
- Como referido, apenas é possível recolher dados de performance na visualização da primeira página, numa *single page application.* O script deveria estar preparado para identificar a suposta mudança de página na framework Angular, utilizada no desenvolvimento da Liberopinion.
- A integrabilidade da plataforma com os *backoffice*s das aplicações, tornaria a análise do desempenho mais cómoda.

A plataforma encontra-se preparada para os *websites* da plataforma Liberopinion. Seria interessante no futuro disponibilizar o serviço para a comunidade Web, de forma a utilizarem a plataforma para a análise de performance dos seus *websites*.

A aplicação da Liberopinion ainda pode ser melhorada em termos de desempenho. Tratando-se de uma *single page application,* os recursos javascript e CSS têm que ser transferidos de uma só vez no primeiro acesso. Constantemente são acrescentados módulos de participação à plataforma Liberopinion portanto, há cada vez mais dados que têm de ser transferidos no primeiro acesso. A certo ponto, este tipo de desenvolvimento não será sustentável, por isso, devem-se procurar soluções para contornar este problema.

As condições de acesso utilizadas nos testes sintéticos para verificar o impacto das técnicas de melhoria implementadas tinham uma largura de banda que corresponde a valores da média mundial, segundo Akamai [41]. Seria interessante que os testes sintéticos tivessem sido executados num contexto extremo, onde a largura de banda teria valores menores, para simular o acesso através de redes móveis. Embora, na rede, a latência seja o fator que tem mais impacto na degradação do desempenho, a largura de banda também é importante [42]. Os dispositivos móveis têm mais limitações no acesso, por causa da latência ao tipo de tecnologia de rede utilizada e o desgaste da sua bateria à medida que aumenta o seu processamento [43]. É importante que a aplicação tenha um desempenho aceitável em quaisquer condições de rede.

Embora tenham sido recolhidos dados de performance e seja possível analisar na plataforma desenvolvida, as condições em que as páginas foram acedidas não foram retiradas conclusões neste trabalho. A sua análise auxilia na identificação das condições que degradam mais o desempenho global dos *websites* avaliados. No futuro esta informação será revelante para identificar problemas e avaliar soluções para otimizar o desempenho das aplicações Web.

REFERÊNCIAS

[1] H. King, «The Web is getting slower», *CNNMoney*, 16-Jun-2015. [Em linha]. Disponível em: http://money.cnn.com/2015/06/16/technology/web-slow-big/index.html. [Acedido: 16-Jul-2015].

[2] M. Csikszentmihalyi, *Flow: The psychology of optimal experience*, Nachdr. New York: Harper [and] Row, 2009.

[3] Amit Singhal, «Using site speed in web search ranking», *Official Google Webmaster Central Blog*, 09-Abr-2010. .

[4] V. Koshy, *Action research for improving practice a practical guide*. London: Thousand Oaks, CA : PCP/Sage Publications, 2005.

[5] N. K. Denzin e Y. S. Lincoln, Eds., *The Sage handbook of qualitative research*, 4th ed. Thousand Oaks: Sage, 2011.

[6] «Manifesto for Agile Software Development». [Em linha]. Disponível em: http://www.agilemanifesto.org/. [Acedido: 18-Out-2015].

[7] A. Cockburn, *Agile Software Development*. Boston: Addison-Wesley Professional, 2001.

[8] Jakob Nielsen, «Response Times: The 3 Important Limits», 01-Jan-1993. [Em linha]. Disponível em: http://www.nngroup.com/articles/response-times-3-important-limits/. [Acedido: 09-Jul-2015].

[9] Kent Alstad, «Spring 2014 State of the Union: Ecommerce Page Speed & Web Performance [INFOGRAPHIC]», *Web Performance Today*, 29-Abr-2014. [Em linha]. Disponível em: http://www.webperformancetoday.com/2014/04/29/spring-2014-state-union-ecommerce-page-speed-web-performance-infographic/. [Acedido: 11-Jul-2015].

[10] L. C. Hogan, *Designing for Performance: Weighing Aesthetics and Speed*, 1 edition. S.l.: O'Reilly Media, 2014.

[11] Jake Brutlag, «Speed Matters», *Research Blog*, 23-Jun-2009. .

[12] Kent Alstad, «4 awesome slides showing how page speed correlates to business metrics at Walmart.com», *Web Performance Today*, 28-Fev-2012. [Em linha]. Disponível em: http://www.webperformancetoday.com/2012/02/28/4-awesome-slides-showing-how-page-speed-correlates-to-business-metrics-at-walmart-com/. [Acedido: 11-Jul-2015].

[13] A. Croll e S. Power, *Complete web monitoring*, 1st ed. Beijing ; Cambridge [Mass.]: O'Reilly, 2009.

[14] T. Barker, *High Performance Responsive Design: Building Faster Sites Across Devices*, 1 edition. Sebastopol, CA: O'Reilly Media, 2014.

[15] «HTTP/1.1: Status Code Definitions». [Em linha]. Disponível em: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. [Acedido: 20-Jul-2015].

[16] Ilya Grigorik, «Constructing the Object Model», *Web Fundamentals*, 01-Abr-2014. [Em linha]. Disponível em: https://developers.google.com/web/fundamentals/. [Acedido: 11-Jul-2015].

[17] Ilya Grigorik, «Critical Rendering Path», *Web Fundamentals*, 01-Abr-2014. [Em linha]. Disponível em: https://developers.google.com/web/fundamentals/. [Acedido: 11-Jul-2015].

[18] Ilya Grigorik, «Render-tree construction, Layout, and Paint», *Web Fundamentals*, 01-Abr-2014. [Em linha]. Disponível em: https://developers.google.com/web/fundamentals/. [Acedido: 11-Jul-2015].

[19] Tammy Everts, «Metrics, metrics everywhere (but where the heck do you start?) | SOASTA», 27-Mai-2015. [Em linha]. Disponível em: http://www.soasta.com/blog/metrics-metrics-everywhere-but-where-the-heck-do-you-

start/?imm_mid=0d2bdb&cmp=em-webops-na-na-newsltr_20150529. [Acedido: 11-Jul-2015].
[20] «Speed Index - WebPagetest Documentation». [Em linha]. Disponível em: https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index. [Acedido: 12-Jul-2015].
[21] Zhiheng Wang e Arvind Jain, «Navigation Timing», 25-Jan-2013. [Em linha]. Disponível em: https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/NavigationTiming/Overview.html. [Acedido: 11-Jul-2015].
[22] «Can I use... Support tables for HTML5, CSS3, etc». [Em linha]. Disponível em: http://caniuse.com/#search=navigation%20timing%20api. [Acedido: 11-Jul-2015].
[23] «Date», *Mozilla Developer Network*. [Em linha]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date. [Acedido: 11-Jul-2015].
[24] Ilya Grigorik, «Measuring Site Speed with Navigation Timing - igvita.com», 04-Abr-2012. [Em linha]. Disponível em: https://www.igvita.com/2012/04/04/measuring-site-speed-with-navigation-timing/. [Acedido: 11-Jul-2015].
[25] «developerWorks Interviews: Tim Berners-Lee», 22-Ago-2006. [Em linha]. Disponível em: http://www.ibm.com/developerworks/podcast/dwi/cm-int082206txt.html. [Acedido: 11-Jul-2015].
[26] Aaron Swartz, «A Brief History of Ajax (Aaron Swartz's Raw Thought)», 22-Dez-2005. [Em linha]. Disponível em: http://www.aaronsw.com/weblog/ajaxhistory. [Acedido: 11-Jul-2015].
[27] Nic Jansma, «How to provide real user monitoring for single-page applications | SOASTA», 20-Mai-2015. [Em linha]. Disponível em: http://www.soasta.com/blog/angularjs-real-user-monitoring-single-page-applications/?imm_mid=0d26f5&cmp=em-webops-na-na-newsltr_20150522%2F. [Acedido: 11-Jul-2015].
[28] Steve Souders, «Comparing RUM & Synthetic Page Load Times | High Performance Web Sites», 14-Nov-2012. .
[29] S. Souders, *High Performance Web Sites: Essential Knowledge for Front-End Engineers*, 1 edition. Farnham: O'Reilly Media, 2007.
[30] «Headless browser», *Wikipedia, the free encyclopedia*. 28-Mai-2015.
[31] «HTML5 Application Cache». [Em linha]. Disponível em: http://www.w3schools.com/html/html5_app_cache.asp. [Acedido: 14-Jul-2015].
[32] Steve Souders, «the Performance Golden Rule | High Performance Web Sites», 10-Fev-2012. .
[33] Ilya Grigorik, «HTTP caching», *Web Fundamentals*, 01-Abr-2014. [Em linha]. Disponível em: https://developers.google.com/web/fundamentals/. [Acedido: 28-Jul-2015].
[34] Andy Davies, «How the Browser Pre-loader Makes Pages Load Faster», 22-Out-2013. [Em linha]. Disponível em: http://andydavies.me/blog/2013/10/22/how-the-browser-pre-loader-makes-pages-load-faster/. [Acedido: 02-Set-2015].
[35] «364315 – Speculatively load referenced files while "real" parsing is blocked on a <script src=> load». [Em linha]. Disponível em: https://bugzilla.mozilla.org/show_bug.cgi?id=364315. [Acedido: 02-Set-2015].
[36] Ilya Grigorik, «Chrome's preloader delivers a ~20% speed improvement! Wait, what's a…», 06-Mai-2013. [Em linha]. Disponível em: https://plus.google.com/+IlyaGrigorik/posts/8AwRUE7wqAE. [Acedido: 02-Set-2015].

[37] S. Souders, *Even Faster Web Sites: Performance Best Practices for Web Developers*, 1 edition. Sebastopol: O'Reilly Media, 2009.
[38] N. C. Zakas, *High Performance JavaScript*, 1st edition. Sebastopol, Calif.: O'Reilly Media, 2010.
[39] «Mobile Analysis in PageSpeed Insights», *Google Developers*. [Em linha]. Disponível em: https://developers.google.com/speed/docs/insights/mobile. [Acedido: 07-Set-2015].
[40] Tammy Everts, «When it comes to delivering the best possible user experience, how fast is fast enough? | SOASTA», 23-Jun-2015. [Em linha]. Disponível em: http://www.soasta.com/blog/website-monitoring-fast-enough-user-experience/?imm_mid=0d4455&cmp=em-webops-na-na-newsltr_20150626. [Acedido: 07-Set-2015].
[41] «State of the Internet Report | Akamai». [Em linha]. Disponível em: https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report/index.jsp. [Acedido: 10-Out-2015].
[42] I. G. on J. 19 e 2012, «Latency: The New Web Performance Bottleneck - igvita.com». [Em linha]. Disponível em: https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/. [Acedido: 12-Out-2015].
[43] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and web performance*, 1 edition. O'Reilly Media, 2013.