CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

OTIMIZANDO DESEMPENHO DE FRONT-END EM WEBSITES PARA HTTP2

PEDRO COLEN CARDOSO

Orientador: Prof. Flávio Coutinho
Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

BELO HORIZONTE
MAIO DE 2015

PEDRO COLEN CARDOSO

OTIMIZANDO DESEMPENHO DE FRONT-END EM WEBSITES PARA HTTP2

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia da Computação do Centro Federal de Educação Tecnológica de Minas Gerais.

Orientador: Flávio Coutinho
Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO
BELO HORIZONTE
MAIO DE 2015

PEDRO COLEN CARDOSO

OTIMIZANDO DESEMPENHO DE FRONT-END EM WEBSITES PARA HTTP2

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia da Computação do Centro Federal de Educação Tecnológica de Minas Gerais.

Trabalho aprovado. Belo Horizonte, 24 de novembro de 2014

Flávio Coutinho
Orientador

Co-Orientador

Professor
Convidado 1

Professor
Convidado 2

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO
BELO HORIZONTE
MAIO DE 2015

Espaço reservado para dedicatória. Inserir seu texto aqui...

Agradecimentos

Inserir seu texto aqui... (esta página é opcional)

"You can't connect the dots looking forward you can only connect them looking backwards. So you have to trust that the dots will somehow connect in your future. You have to trust in something: your gut, destiny, life, karma, whatever. Because believing that the dots will connect down the road will give you the confidence to follow your heart, even when it leads you off the well worn path."(Steve Jobs)

Resumo

Abstract

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Algoritmos

Lista de Abreviaturas e Siglas

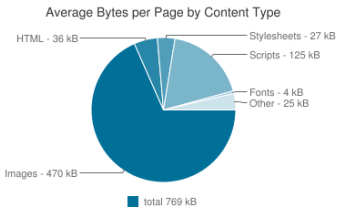
AJAX	Asynchronous JavaScript and XML
IETF	Internet Engineering Task Force
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTP-WG	HTTP Working Group
kb	Kilobytes
MIME	Multi-Purpose Internet Mail Extensions
ms	Milisegundo
RFC	Request for Comments
TCP	Transmission Control Protocol
XML	Extensible Markup Language

Sumário

1 Introdução

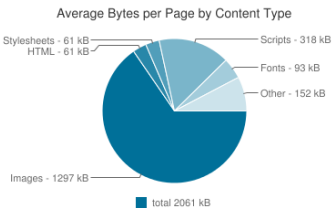
Desde que o modelo atual da Internet foi proposto pelo cientista e pesquisador britânico Tim Bernes-Lee em 1989 (CONNOLLY, 2000), as páginas *web* vêm mudando de maneira acelerada como pode ser percebido observando os gráficos nas Figura 1 e Figura 2, gerados com a ajuda do *website* HTTP Archieve. O primeiro foi gerado com dados de 15 de Abril de 2011 e o segundo com dados de 15 de Abril de 2015 e os dois mostram a média de *bytes* por página por tipo de conteúdo nas páginas *web*. Mas a informação mais relevante está localizado na parte debaixo do gráfico e mostra o tamanho médio de uma página *web* nas respectivas dadas.

Figura 1 – Média de Bytes por Página por Tipo de Conteúdo em 2011



Fonte: Archive (2015a)

Figura 2 – Média de Bytes por Página por Tipo de Conteúdo em 2015



Fonte: Archive (2015b)

Nos últimos 4 anos o tamanho médio de uma página *web* passou de 769kb para 2061kb, um impressionante aumento de 168%! Apesar dessa grande mudança no tamanho das páginas (e consequentemente dos *websites*) a maneira como *websites* são entregues dos servidores para os clientes não sofreu nenhuma alteração desde 1999, ano de lançamento da RFC 2616 que especificou o HTTP/1.1 (GROUP, 1999). Como explicado por (TANENBAUM, 2011), o HTTP é um simples protocolo de pedidos e respostas que roda em cima do protocolo TCP. O HTTP ficou famoso por ser fácil de entender e implementar e ao mesmo tempo cumprir sua função com um bom desempenho. Contudo, o aumento no tamanho dos *websites* começou a fazer com que o tempo de resposta das páginas *web* ficasse muito grande, e como mudanças no

HTTP não eram possíveis, os desenvolvedores passaram a ter de criar outras formas de resolver esse problema.

Técnicas de otimização de desempenho passaram a ser estudadas e implementadas por muitas empresas que queriam ter seus *websites* entregues mais rapidamente à seus clientes. por muitos anos a grande maioria dessas empresas focou seus esforços em otimizações para o *back-end*, principalmente para os seus servidores, o que hoje em dia pode ser considerado um erro. Como explicado por (SOUDERS, 2007) no que ele chamou de "Regra de Ouro do Desempenho":

> "Apenas 10-20% do tempo de resposta do usuário final são gastos baixando o documento HTML. Os outros 80-90% são gastos baixando todos os componentes da página"

Dessa forma ficou claro que técnicas de otimização de desempenho para o *front-end* dos *websites* deveriam se tornar prioridade quando procura-se melhorar o tempo de resposta para o usuário final. Para entender o quão importante esse tempo de resposta se tornou para empresas que dependem da Internet para sobreviver, basta observar os dados da tabela Tabela 1 expostos por Steve Souders na conferência Google I/O de 2009:

Tabela 1 – Impacto do desempenho de *website* na receita.

Empresa	**Piora no tempo de resposta**	**Consequencia**
Google Inc.	+500ms	-20% de tráfego
Yahoo Inc.	+400ms	-5% à -9% de tráfego
Amazon.com Inc.	a cada +100ms	-1% de vendas

Fonte: (SOUDERS, 2009b)

Steve Souders tornou-se um grande evangelizador da área de otimização de desempenho de *front-end* de *websites*. Em seus livros, *High Performance Websites* (SOUDERS, 2007) e *Even Faster Websites* (SOUDERS, 2009a) ele ensina técnicas de como tornar *websites* mais rápidos focando nos componentes das páginas. E em 2012 ele lançou seu terceiro livro, *Web Performance Daybook* (SOUDERS, 2012), como um guia para desenvolvedores que trabalham com otimização de desempenho de *websites*.

1.1 Motivação

Após mais de 15 anos sem mudanças, o protocolo HTTP (finalmente) receberá uma atualização. A nova versão do protocolo, chamada de HTTP2, teve sua especificação aprovada no dia 11 de Fevereiro de 2015, (GROUP, 2015), e deverá começar a ser implantada a partir de 2016. Muitas mudanças foram feitas com o objetivo de melhorar

o desempenho e a segurança da Internet. Além disso o HTTP2 foi desenvolvido para ser compatível com seus versões anteriores, não sendo necessárias mudanças em servidores e aplicações antigos para funcionar em cima do novo protocolo.

Com as novas funcionalidades do HTTP2 a caminho algumas coisas devem mudar na área de otimização de desempenho de *websites*. Como pode ser percebido em (STENBERG, 2014), o HTTP2 foi desenvolvido para melhorar o desempenho de todos os *websites* e aplicações *web*, e não apenas dos poucos que podem aplicar técnicas de otimização. Então fica difícil de prever o resultado da aplicação de técnicas desenvolvidas para os protocolos HTTP/1.0 e HTTP/1.1. Acredita-se que algumas das técnicas antigas podem não apenas não melhorar o desempenho dos *websites* como podem acabar piorando o tempo de resposta para o usuário final.

No decorrer dos próximos anos o HTTP2 deve seguir o mesmo caminho do HTTP/1.1 e se tornar o protocolo mais utilizado da Internet. Apesar de todo o esforço do HTTPbis (grupo responsável por desenvolver a especificação do HTTP2) em desenvolver um protocolo que garanta o melhor desempenho de *websites* e aplicações sempre é possível ser mais rápido se as medidas certas forem tomadas.

1.2 Objetivos

Este trabalho tem como objetivo analisar o comportamento de técnicas de otimização de desempenho de *websites* desenvolvidas para os protocolos HTTP/1.0 e HTTP/1.1 quando aplicadas em cima do protocolo HTTP2 e, se necessário, propor técnicas especificas para o novo protocolo.

Para realização do objetivo principal os seguintes objetivos específicos foram determinados:

1. Fazer uma análise comparativa das versões do protocolo HTTP
2. Avaliar os ganhos de desempenho das técnicas propostas por Steve Souders ao aplicá-las ao HTTP2
3. Se necessário, propor novas técnicas de otimização de desempenho de *websites* específicas para o HTTP2

2 Trabalhos Relacionados

3 Fundamentação Teórica

Para se obter grandes resultados na otimização de *front-end* de *websites* é essencial ter conhecimento de como o protocolo HTTP evoluiu e como ele funciona assim é possível encontrar seus pontos fracos e trabalhar em maneiras para minimizá-los. Além disso é necessário compreender o que são as chamadas assíncronas de JavaScript e XML, conhecidas como AJAX, e porque elas são tão utilizadas em *websites* e aplicações para a chamada Web 2.0.

3.1 O Protocolo HTTP

3.2 História

Em 1989 Tim Berners-Lee propôs uma rede de computadores global para ajudar a comunidade científica a compartilhar o conhecimento gerado em diferentes partes do mundo, acelerando assim o desenvolvimento tecnológico. Ao final de 1990 o cientista do CERN já tinha desenvolvido em um computador de seu laboratório o primeiro navegador *web* (chamado de WorldWideWeb), uma linguagem de marcação de hipertextos (o HTML) e um protocolo para a transferência de hipertextos (o HTTP).

O HTTP recebeu sua primeira documentação oficial em 1991, quando passou a ser chamado de HTTP/0.9. Como esclarece (GRIGORIK, 2013), todas as versões anteriores passaram a ser consideradas iterações com o objetivo de se chegar a versão HTTP/0.9, então todas elas acabaram recebendo o rótulo de HTTP/0.9.

Essa primeira versão do protocolo era extremamente simples. A ideia é explicada por (GRIGORIK, 2013) da seguinte maneira:

- A requisição do cliente é uma cadeia simples de caracteres ASCII
- A resposta do servidor é uma torrente de caracteres ASCII
- A resposta do servidor é um HTML
- A conexão é fechada após o transferência do documento

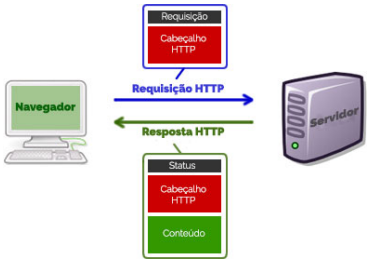
Com o passar dos anos a *World Wide Web* de Tim Berners-Lee cresceu rapidamente e isso fez com que o uso do HTTP aumentasse muito em pouco tempo. Viu-se a necessidade de um protocolo mais robusto e estruturado, mas que mantivesse a simplicidade do HTTP/0.9, pois esse era o segredo por trás de seu sucesso. Então o IETF

passou a coordenar a criação de especificações para o HTTP e criou o HTTP-WG que tinha como função criar definição das versões seguintes do protocolo. Então, em 1996 foi definida a versão HTTP/1.0 (GROUP, 1996), em 1999 a versão HTTP/1.1 (GROUP, 1999) e em 2015 foi aprovada a especificação para a versão HTTP2 (GROUP, 2015).

3.3 Visão Geral

Como dito por (TANENBAUM, 2011), o HTTP é um simples protocolo de requisições e respostas que normalmente funciona em cima do protocolo TCP. As requisições e respostas são compostas por um cabeçalho HTTP e um conteúdo, e são enviadas do cliente para o servidor. O modelo, ilustrado na Figura 3, é bem simples e direto, fácil de ser replicado.

Figura 3 – Visão Geral do Protocolo HTTP



Fonte: Adaptado de Saenz (2015)

O protocolo foi desenvolvido para funcionar na camada de Aplicação do protocolo TCP, conectando as ações do usuário a camada de Apresentação. Contudo de acordo com (TANENBAUM, 2011), de certa forma o HTTP se transformou em um protocolo da camada de Transporte, criando uma maneira de processos se comunicarem através de diferentes redes. Hoje em dia não são apenas os navegadores *web* que utilizam o protocolo HTTP para se comunicar com servidores, tocadores de mídias, anti-vírus, programas de fotos, dentre outros tipos de aplicações utilizam o HTTP para recolher informações de servidores de maneira simples e rápida.

Os cabeçalhos HTTP definem características desejadas ou esperadas pelas aplicações e servidores, como tipo de codificação de caracteres ou tipo de compressão dos dados. Existem várias etiquetas padrões que podem ser utilizadas nos cabeçalhos e a desenvolvedores podem ainda criar etiquetas próprias para serem utilizadas dentro das aplicações (caso um cliente ou servidor receba uma etiqueta que não reconhece ele simplesmente a ignora). No Quadro 1 são apresentadas algumas das etiquetas mais utilizadas, mas existem muitas outras que não foram citadas e que podem variar com a versão do protocolo utilizada por ambas as partes envolvidas na troca de dados.

Quadro 1 – Etiquetas para cabeçalhos HTTP.

Etiqueta	**Tipo**	**Conteúdo**
Accept	Requisição	Tipo de páginas que o cliente suporta
Accept-Encoding	Requisição	Tipo de codificação que o cliente suporta
If-Modified-Since	Requisição	Data e hora para checar atualidade do conteúdo
Authorization	Requisição	Uma lista de credencias do cliente
Cookie	Requisição	Cookie definido previamente enviado para o servidor
Content-Encoding	Resposta	Como o conteúdo foi codificado (ex. *gzip*)
Content-Length	Resposta	Tamanho da página em *bytes*
Content-Type	Resposta	Tipo de *MIME* da página
Last-Modified	Resposta	Data e hora que a página foi modificada pela última vez
Expires	Resposta	Data e hora quando a página deixa de ser válida
Cache-Control	Both	Diretiva de como tratar *cache*
ETag	Both	Etiqueta para o conteúdo da página
Upgrade	Both	O protocolo para o qual o cliente deseja alterar

Fonte: Adaptado de Tanenbaum (2011)

O conteúdo de uma resposta HTTP pode variar de formato (HTML, CSS, JavaScript, etc), a definição desse formato é feita com a etiqueta *Content-Type* enviada no cabeçalho de resposta. O conteúdo é a maior parte de uma resposta HTTP e os desenvolvedores devem se esforçar para deixá-lo o menor possível, possibilitando assim que a comunicação de dados seja mais rápida.

Por rodar em cima do protocolo TCP o HTTP precisa que uma conexão TCP seja aberta para poder realizar a troca de dados entre o cliente e o servidor. Como essa conexão é gerenciada vai depender da versão do protocolo. Após a abertura da conexão a requisição pode ser enviada. Na primeira linha da requisição são definidas a versão do protocolo e a operação que será realizada. Apesar de ter sido criado apenas para recuperar páginas *web* de um servidor, o HTTP foi intencionalmente desenvolvido de forma genérica, possibilitando a extensibilidade do seu uso. Sendo assim o protocolo suporta diferentes operações, chamadas de métodos, além da tradicional requisição de páginas *web*. A lista completa de métodos com suas descrições pode ser vista no Quadro 2.

Destes métodos vale a pena enfatizar os métodos GET e POST que são os mais utilizados pelos navegadores *web* e os que serão mais utilizados neste trabalho. O método GET é utilizado para recuperar informações do servidor e o método POST para enviar informações para o servidor.

Sempre que uma requisição é enviada ela recebe uma resposta, mesmo que a requisição falhe. Na primeira linha do cabeçalho de resposta se encontra o código do estado da resposta em formato numérico de três dígitos. O primeiro digito deste código define a qual sub-grupo ele pertence. O Quadro 3 mostra os sub-grupos existentes

Quadro 2 – Métodos HTTP.

Método	**Descrição**
GET	Ler página *web*
HEAD	Ler cabeçalho de página *web*
POST	Anexar à página *web*
PUT	Armazenar página *web*
DELETE	Remover página *web*
TRACE	Imprimir requisição de entrada
CONNECT	Conectar através de um *proxy*
OPTIONS	Listar opções para uma página *web*

Fonte: Adaptado de Tanenbaum (2011)

e o significado de cada um deles. Cada um destes sub-grupos possui vários códigos com diferentes significados e com a evolução do protocolo mais códigos foram sendo inseridos para lidar com necessidades especificas.

Quadro 3 – Códigos de estado HTTP.

Código	**Significado**	**Exemplo**
1xx	Informação	100 = servidor concorda em lidar com requisição do cliente
2xx	Sucesso	200 = sucesso na requisição; 204 = nenhum conteúdo presente
3xx	Redirecionamento	301 = página foi movida; 304 = *cache* ainda é válida
4xx	Erro do cliente	403 = página proibida; 404 = página não encontrada
5xx	Erro do servidor	500 = erro interno de servidor; 503 = tente novamente mais tarde

Fonte: Adaptado de Tanenbaum (2011)

A *cache* é uma característica importante do HTTP. O protocolo foi construído com suporte integrado para lidar com este importante requisito de desempenho. Os clientes e os servidores conseguem gerenciar *caches* com a ajuda dos cabeçalhos de requisição e resposta, e o tamanho da *cache* é definido pelo navegador. O problema destas memórias locais é saber o momento de utilizar os dados armazenados nelas ou de pedir novos dados ao servidor. Para isso existem variadas técnicas, como as propostas por (SOUDERS, 2007), que devem ser analisadas e utilizadas de acordo com a situação.

As versões do protocolo HTTP foram aprimorando falhas identificadas quando ele passou a ser utilizado em larga escala. Mas apesar das mudanças a essência do protocolo sempre continuou a mesma: um simples protocolo de requisição e resposta. Nas próximas seções será apresentada uma comparação entre as versões do HTTP, com ênfase nas mudanças que ajudaram a melhorar a performance dos *websites* e aplicações *web*.

3.4 HTTP/1.0 VS HTTP/1.1

Com a popularização da Internet no inicio da década de 1990 o uso do HTTP estava crescendo muito rápido. Com isso o IETF teve que se apressar para criar um documento de consulta para desenvolvedores que queriam utilizar o protocolo em suas aplicações. Sendo assim, apesar de todo o debate por trás da (GROUP, 1996), aprovada em 1996, o documento apenas explicou os usos comuns do HTTP/1.0, mas não chegou a definir padrões de como utiliza-lo, como explica (**??**). Por isso logo após a sua aprovação o HTTP-WG já começou a trabalhar na (GROUP, 1999), para poder corrigir os erros existentes no HTTP/1.0 com a criação de uma nova versão, o HTTP/1.1.

Várias funcionalidades importantes foram adicionadas no HTTP/1.1, e existiu uma preocupação muito grande com a compatibilidade entre as versões do protocolo, afinal de contas o HTTP/1.0 já era amplamente utilizado e não podia se esperar que todos os*websites* e aplicações se adaptassem de uma hora para outra. Esse fato também levou o HTTP-WG a criar um protocolo que suportasse futuras mudanças (lembrando que no HTTP todas as etiquetas que um cliente ou servidor não reconhecem são simplesmente ignoradas). Pensando na extensibilidade do protocolo as primeiras mudanças no protocolo HTTP/1.1 foram a criação de duas novas etiquetas para cabeçalhos, *Upgrade* - uma maneira do cliente informar qual versão do protocolo ele suporta - e *Via* - que define uma lista dos protocolos suportados pelos clientes ao longo do caminho de uma transmissão.

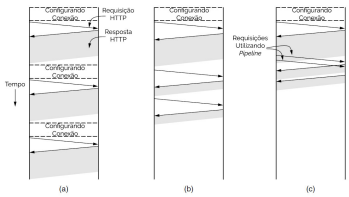
Como dito anteriormente o protocolo HTTP foi construído com suporte integrado para *cache*. Mas o mecanismo de *cache* do HTTP/1.0 era muito simples e não permitia que o cliente ou o servidor definissem instruções diretas de como a memória deveria ser utilizada. O HTTP/1.1 tentou corrigir esse problema com a criação de novas etiquetas para cabeçalhos. A primeira delas é a *ETag*, que define uma cadeia de caracteres única para um arquivo. Além do próprio conteúdo, essa cadeia utiliza a data e a hora da última modificação no arquivo, logo pode ser utilizada para verificar se dois arquivos são idênticos. O HTTP/1.1 também definiu novas etiquetas condicionais para complementar a já existente *If-Modified-Since*, as *If-None-Match*, *If-Match* e *If-Unmodified-Since*, que podem ser usada para verificar se arquivos em *cache* estão atualizados ou não. Uma das mudanças mais significativas no mecanismo de *cache*, foi a etiqueta *Cache-Control* que permiti definir novas diretrizes para o uso da *cache*, como tempo de expiração relativos e arquivos que não devem ser armazenados.

O HTTP/1.0 tinha muitos problemas em gerenciar a largura de banda, por exemplo não era possível enviar partes de arquivos, logo, mesmo se o cliente não precisasse de um arquivo inteiro, ele teria de recebe-lo. No HTTP/1.1 foram então criados a etiqueta *Range*, o tipo *MIME multipat/byteranges* e o tipo de compressão *Chuncked* para que cliente e servidores pudessem trocar mensagens com partes de

arquivos. Para complementar esse novo mecanismo foi incluído um novo código de resposta, 100, para informar um cliente que o corpo de sua requisição deve ser enviado. Além de poder enviar partes de arquivos o HTTP/1.1 se preocupou em garantir a compressão dos dados durante todo o caminho da transmissão, então foi incluída a etiqueta *Transfer-Encoding*, que complementa a *Content-Encoding* indicando quando codificação foi utilizada de um ponto para o outro.

O modelo original do protocolo HTTP utilizava uma conexão TCP para cada transmissão. Esse processo era extremamente danoso para o desempenho de *websites* e aplicações, pois se gastava muito tempo na criação e configuração de novas conexões e nos primeiros momentos da iniciais da conexão (quando ela é mais lenta por definição). Para corrigir esse problema o HTTP/1.1 define conexões persistentes como seu padrão. Conexões persistentes permitem que clientes e servidores assumam que uma conexão TCP continuará aberta após a transmissão de dados, e que esta pode ser utilizada para uma nova transmissão. Além disso foi definido que o HTTP/1.1 utilizaria *pipeline*, isto que dizer que clientes não precisam aguarda a resposta de uma requisição para mandarem uma nova requisição, como era o padrão do HTTP/1.0. Os ganhos com essas novas técnicas podem ser notados na **??**.

Figura 4 – HTTP com (a) múltiplas conexões e requisições sequenciais. (b) Uma conexão persistente e requisições sequenciais. (c) Uma conexão persistente e requisições em pipeline



Fonte: Adaptado de Tanenbaum (2011)

Uma funcionalidade desejada no HTTP/1.1 era a de poder fazer requisições para servidores outros além do da página principal sendo acessada. Isso possibilita que desenvolvedores hospedem arquivos CSS e JavaScript em um servidor e imagens em outro por exemplo. Isso se tornou possível com a adição da etiqueta *Host*, onde o cliente pode definir que é o caminho do servidor que será utilizado na requisição. Caso a etiqueta *Host* não esteja definida no cabeçalho é assumido que o caminho do servidor é o caminho da página principal.

Como conclui (**??**): os protocolos HTTP/1.0 e HTTP/1.1 diferem em diversas maneiras. Enquanto muitas dessas mudanças têm o objetivo de melhorar o HTTP, a descrição do protocolo mais do que triplicou de tamanho, e muitas dessas funcionalidades foram introduzidas sem testes em ambientes reais. Esse aumento de complexidade causou muito trabalho para os desenvolvedores de clientes e servidores *web*.

No BLA encontra-se um resumo das mudanças introduzidos no protocolo HTTP/1.1. Observa-se que nem todas as mudanças mostradas no BLZ foram explicadas anteriormente, mas servem para ilustrar que existem ainda outras diferenças entre as versões 1.0 e 1.1 do protocolo que para os fins deste trabalho elas não são relevantes.

Quadro 4 – Cronograma de atividades.

Cabeçalhos	
Etiqueta	**Descrição**
*Accept-Encoding**	Lista de codificações aceitas
Age	O tempo que o conteúdo está salvo no *proxy em segundos*
Cache-Control	Notifica todos os mecanismos de *cache* do cliente ao servidor se o conteúdo deve ser salvo
Connection	Controla a conexão atual
Content-MD5	Codificação binária de base-64 para verificar conteúdo da resposta
Etag	Identificador único de um conteúdo
Host	Indica o endereço e a porta do servidor que deve ser utilizado pela mensagem
If-Match	Realize a ação requisitada se, e somente se, o conteúdo do cliente é igual ao conteúdo do servidor
If-None-Match	Retorna código 304 se o conteúdo não foi modificado
If-Range	Se o conteúdo não foi modificado, envie a parte solicitada, se não, envie o conteúdo novo
If-Unmodified-Since	Envie o conteúdo se, e somente se, ele não foi modificado na data esperada
Proxy-Authentication	Pede uma requisição de autenticação de um *proxy*
Proxy-Autorization	Credenciais de autorização para se conectar a um *proxy*
Range	Faz a requisição de apenas uma parte de um conteúdo
Trailer	Indica que o grupo de cabeçalhos está presente em uma mensagem
Transfer-Encoding	A forma de codificação usada para se transferir o conteúdo para o usuário
Upgrade	Pede para o servidor atualizar para outro protocolo
Vary	Informa o cliente sobre *proxies* pelos quais a resposta passou
Via	Informa o servidor de *proxies* pelos quais a requisição passou
Warning	Mensagem genérica de cuidado para possíveis problemas no corpo da mensagem
WWW-Authenticate	Indica tipo de autenticação que deve ser utilizada para acessar entidade requerida
Métodos	
Método	**Descrição**
OPTIONS	Requer informações sobre os recursos que o servidor suporta
Estados**	
Código	**Descrição**
100	Confirma que o servidor recebeu o cabeçalho de requisição e que o cliente deve continuar a enviar a mensagem desejada
206	O servidor está entregando apenas uma parte de um conteúdo por causa da etiqueta de *Range* na requisição do cliente
300	Indica as multiplas opções disponíveis para o cliente
409	Indica que a requisição não pôde prosseguir por causa de um conflito
410	Indica que o conteúdo requisitado não está mais disponível e não estará disponível no futuro
Diretivas	
Diretiva	**Descrição**
chuncked	Utilizado para envie de conteúdo em partes
max-age	Determina qual é o tempo máximo que um conteúdo deve ficar salvo em *cache*
no-store	Indica que o conteúdo não deve ser salvo em *cache*
no-transform	Indica que o conteúdo não deve ser modificado por *proxies*
private	Indica que o conteúdo não deve ser acessado sem autenticação
Tipos de MIME	
Tipo	**Descrição**
multipart/byteranges	Indica que o conteúdo que está sendo enviado é apenas uma parte de um todo
Funcionalidades	
Nome	**Descrição**
Content negotitation	Escolhe a melhor representação disponível para um conteúdo
Persistent connection	Após o termino de uma requisição HTTP a conexão continua aberta e pode ser utilizada por outras requisições
Pipeline	O cliente não precisa esperar que a resposta de uma requisição retorne antes de enviar outra requisição

4 Metodologia

Inserir seu texto aqui...

4.1 Delineamento da pesquisa

Inserir seu texto aqui...

4.2 Coleta de dados

Inserir seu texto aqui...

5 Análise de Resultados

Inserir seu texto aqui...

5.1 Situação atual

Inserir seu texto aqui...

5.2 Análise dos dados coletados

Inserir seu texto aqui...

6 Conclusão

Referências

ARCHIVE, H. **Http Archive Abril 2011 Query**. 2015. Disponível em: <http://httparchive.org/interesting.php?a=All&l=Apr%2015%202011>. Citado na página 1.

ARCHIVE, H. **Http Archive Abril 2015 Query**. 2015. Disponível em: <http://httparchive.org/interesting.php?a=All&l=Apr%2015%202015>. Citado na página 1.

CONNOLLY, D. **The birth of the web**. 2000. Disponível em: <http://www.w3.org/History.html>. Citado na página 1.

GRIGORIK, I. **High Performance Browser Networking**. 1st. ed. [S.l.]: O'Reilly, 2013. Citado na página 5.

GROUP, H. W. **Hypertext Transfer Protocol version 2**. 2015. Disponível em: <https://tools.ietf.org/html/draft-ietf-httpbis-http2-17>. Citado 2 vezes nas páginas 2 e 6.

GROUP, N. W. **RFC 1945**. 1996. Disponível em: <http://tools.ietf.org/html/rfc1945>. Citado na página 6.

GROUP, N. W. **RFC 2616**. 1999. Disponível em: <http://tools.ietf.org/html/rfc2616>. Citado 2 vezes nas páginas 1 e 6.

SAENZ, J. **Building Web Apps with Go**. [s.n.], 2015. Disponível em: <https://www.gitbook.com/book/codegangsta/building-web-apps-with-go/details>. Citado na página 6.

SOUDERS, S. **High Performance Web Sites**. 1st. ed. [S.l.: s.n.], 2007. Citado 2 vezes nas páginas 2 e 8.

SOUDERS, S. **Even Faster Web Sites**. 1st. ed. [S.l.: s.n.], 2009. Citado na página 2.

SOUDERS, S. Even faster websites. In: . [s.n.], 2009. Disponível em: <https://www.youtube.com/watch?feature=player_embedded&v=aJGC0JSlpPE>. Citado na página 2.

SOUDERS, S. **Web Performance Daybook**. 1st. ed. [S.l.: s.n.], 2012. Two. Citado na página 2.

STENBERG, D. Http2 explained. **ACM SIGCOMM Computer Communication Review**, 2014. Disponível em: <http://daniel.haxx.se/http2>. Citado na página 3.

TANENBAUM, D. J. W. A. S. **Computer Networks**. 5th. ed. [S.l.: s.n.], 2011. Citado 4 vezes nas páginas 1, 6, 7 e 8.

Apêndices

APÊNDICE A – Nome do Apêndice

Inserir seu texto aqui...

APÊNDICE B – Nome do Apêndice

Inserir seu texto aqui...

Anexos

ANEXO A – Nome do Anexo

Inserir seu texto aqui...

ANEXO B – Nome do Anexo

Inserir seu texto aqui...