Licenciatura Engª Informática
18 de Setembro de 2014

Relatório do Trabalho Final de Curso

Trabalho realizado sob a orientação de:
Professor José Faísca

Discentes: **Afonso Soromenho| 21001181**
João Mendes | 21003781

A year ago if someone asked you "how do I stream binary data / audio / video / files to Javascript?" the answer would have been "Flash" or "no" (or "Java applets")

BinaryJS

After over 20 years of stateless-web based on the stateless request-response paradigm, we finally have web applications with real-time, two-way connections.

NodeJs

ÍNDICE

ABSTRACT

The project consist in the study and implementation of a video streaming and upload web solution, based on web sockets and html5 with the aim of explore the capacity to establish a communication between the browser(client) and the video server, and evaluate its streaming functionality.

Web sockets is a technology that allows a bi-directional communication over a single TCP socket.
This technology was projected to be used in browsers and web-servers that support HTML5, but it can also be used by any client and application servers.

HTML5 it's the fifth version of HTML, this new version brings new functionalities and resources, that in the past were only possible by the means of other technologies. In it's essence it want's to improve the HTML language with the support for the new multimedia types.

To develop this project we used the Node.js platform, from the server side we used the BinaryJS module for multiple streams over one single websocket connection in real time and we used the Express module for the applicational Web. From the client side it's used HTML5, CSS and JavaScript.

With this project we achieved to implement a video streaming solution on the web, not having time to implement on Samsung smart tv platform, remaining only with a simple but functional prototype that proves the possibility to use the web sockets technology on smart tv's

RESUMO

O projecto consiste no estudo e implementação de uma solução de vídeo upload e streaming na web baseado em WebSockets e HTML5, com intuito de explorar a capacidade de estabelecer uma sessão de comunicação interativa entre o cliente Browser e o servidor de vídeo, e avaliar a funcionalidade de streaming.

WebSocket é uma tecnologia que permite a comunicação bidirecional sobre um único socket TCP Transmission Control Protocol). Projetado para ser executado em browsers e servidores web que suportem o HTML5, mas pode ser usado por qualquer cliente ou servidor de aplicações.

HTML5 é a quinta versão da linguagem HTML. Esta nova versão traz consigo novas funcionalidades e recursos, antes só possíveis por meio de outras tecnologias. Na sua essência, pretende melhorar a linguagem HTML com o suporte para os mais recentes tipos de multimédia.

Para o desenvolvimento deste projecto Foi utilizada a plataforma Node.js, bem como alguns módulos desta plataforma. Do lado servidor o módulo BinaryJS para múltiplos streams sobre uma única conexão WebSocket em tempo real e o módulo Express como a framework aplicacional Web. Do lado do cliente, HTML5, CSS e Javascript.

Com este projecto conseguimos implementar uma solução que faz stream de vídeo na web, não tendo tempo para implementar na Samsung Smart TV, ficando só com um protótipo simples, mas funcional, que prova a possibilidade de utilizar as WebSockets para Smart TV (criar ligação, envio de uma imagem, recepção, e exibição da mesma).

INTRODUÇÃO

A submissão efectuada deste projecto adveio do facto de termos estado a trabalhar com o ambiente *Samsung Smart TV* durante todo o primeiro semestre e termos conhecido o ambiente de trabalho e a sua implementação, bem como a implementação de AJAX para ligações externas à aplicação.

Em termos técnicos optámos por usar JavaScript que continua a ser a linguagem mais usada para web, uma vez que é facilmente interpretada pelo browser que a processa. Para construção do interface usámos HTML5 e CSS3.

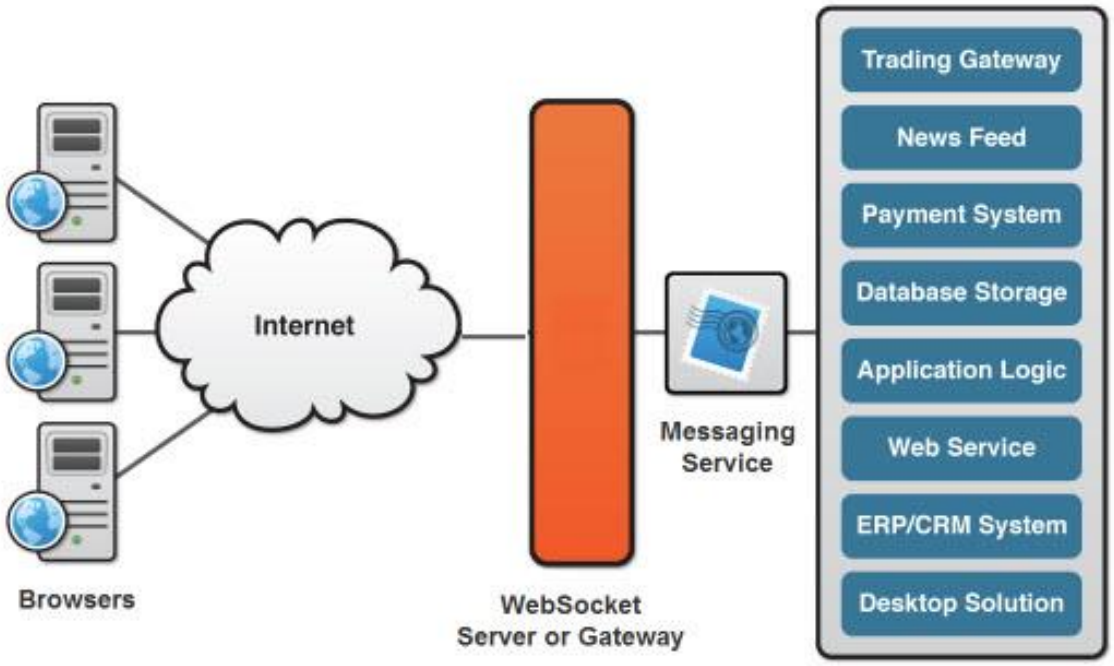
Outro aspeto que tivemos em conta foi o estudo dos formatos de vídeo para que todos os browsers suportassem o mesmo formato e assim o pudessem reproduzir.

O PROJETO *WEBSOCKETS*

A especificação HTML5 *WebSockets* define uma API que permite que páginas web utilizem o protocolo *WebSockets* para comunicação bidireccional com um *host* remoto. A especificação introduz a interface *WebSocket* e define um canal de comunicação full-duplex, que funciona através de uma única *socket* na web. *WebSockets* HTML5 oferecem uma enorme redução no tráfego de rede desnecessário e latência.

HTML5 *WebSockets* têm em consideração *proxies* e *firewalls*, fazendo possível *streaming* através de qualquer conexão, e com a capacidade de suportar a comunicação *downstream* e *upstream* através de uma única conexão. Aplicações baseadas em HTML5 *WebSockets* colocam menos carga em servidores, permitindo que as máquinas existentes suportem conexões simultâneas.

A figura seguinte mostra uma arquitectura básica baseada em WebSocket na qual os *browsers* usam conecção *full-duplex WebSocket*, para comunicação directa com *hosts* remotos.



PROBLEMA

Encontrámos uma vulnerabilidade aquando o desenvolvimento para *Samsung Smart TV*. Os pedidos que hoje em dia são feitos por as TV's (e por maior parte dos serviços online) a qualquer servidor externo são feitos por XMLHttpRequest (XHR), pedidos Ajax usando a tecnologia fornecida pelo jQuery.

Estes pedidos são lentos, e a probabilidade de falhar é elevada. Actualmente usa-se o protocolo *RESTFUL*, permitindo a utilização de JSON o que poderá acelerar o processo, mas mesmo assim continua a falhar, pois caso haja um problema com o servidor, ou ligação, ou (no caso do POST) os parâmetros estejam errados, é necessário repetir todo o processo do pedido.

Além disso os pedidos são pesados, o que em dispositivos com um nível de processamento mais baixo, pode fazer com que o dispositivo não execute na sua totalidade ou execute mal.

SOLUÇÃO

Para solucionar este problema pensámos em usar HTML5 por ser um standart e por a tecnologia permite fazer design responsivo que se adapta a qualquer dispositivo. Usámos também Javascript sem qualquer tipo de plugins.

Utilizámos *WebSockets* para ligações ao *back-end*. Backend foi realizado em NodeJs, BinaryJs para transmissão de dados entre cliente e servidor.

Para reprodução do vídeo usámos formatos standart tais como MP4, WebM e OGG.

O QUE É UM *WEBSOCKET*

WebSocket é um protocolo que cria canais de comunicação *full-duplex* numa conexão TCP. O protocolo *WebSocket* foi estandardizado pelo IETF como RFC 6455 em 2011, e a API do *WebSocket* na Web IDL está a ser estandardizado pela W3C.

O *WebSocket* foi desenhado para ser implementado em web *servers* e *web browsers*, mas pode ser implementado em qualquer aplicação cliente ou de servidor. A única relação que o protocolo WebSocket tem com o HTTP é o HandShake que é interpretado como um Upgrade request pelos servidores HTTP.

O protocolo *WebSocket* torna possível mais interacção entre o *browser* e um website facilitando a criação de conteúdo em tempo real, isto é possível fornecendo uma forma estandardizada para servidor enviar conteúdo para o browser sem que este tenha sido pedido pelo cliente, e permitindo que seja possível a troca de mensagens cliente-servidor mantendo a conexão aberta, desta forma uma "conversa" bidireccional pode ser mantida entre o cliente e o servidor.

As comunicações em *WebSockets* são feitas por TCP e pela porta 80, que beneficia ambientes que bloqueiem as conecções utilizando *firewalls*.

O protocolo *WebSocket* é suportado em todos os browsers modernos incluindo Internet Explorer, Firefox, Safari, Opera e Google Chrome.

LIMITAÇÕES

Cada nova tecnologia tem um novo conjunto de problemas. No caso do *WebSocket*, é a compatibilidade com os servidores proxy que faz a mediação das conexões HTTP na maioria das redes corporativas. O protocolo *WebSocket* usa o sistema de upgrade HTTP (que normalmente é usado para HTTP/SSL) para fazer "upgrade" de uma conexão HTTP para uma conexão *WebSocket*. Alguns servidores proxy não aceitam isso e poderão fechar a conexão. Assim, mesmo se um determinado cliente usar o protocolo *WebSocket*, talvez não seja possível estabelecer uma conexão.

A IMPLEMENTAÇÃO DE *WEBSOCKETS*

O protocolo *WebSocket* foi projectado para funcionar bem com a infra-estrutura da Web existente. Como parte deste princípio de design, a especificação do protocolo define que a conexão *WebSocket* começa como uma conexão HTTP, garantindo total compatibilidade com o mundo pré-*WebSocket*. A troca do protocolo de HTTP para *WebSocket* é referido como um *handshake WebSocket*.

O navegador envia um pedido ao servidor, o que indica que ele quer mudar os protocolos de HTTP para *WebSocket* e com a introdução de uma interface sucinta, os *developers* podem deixar de utilizar técnicas como *Long-polling* e *Forever Frames*, e assim reduzir latência

Para conectar-se a um *end-point*, criamos uma instância de *WebSocket*, fornecemos ao novo objecto um URL que representa o *end-point* a que nos queremos conectar. Note-se que os prefixos ws:// e wss:// representam *WebSocket* e *Secure WebSocket* respectivamente.

```
var myWebSocket = new WebSocket("ws://www.example.com");
```

Podemos associar uma serie de *event listeners* para tratar cada pedido do utilizador.

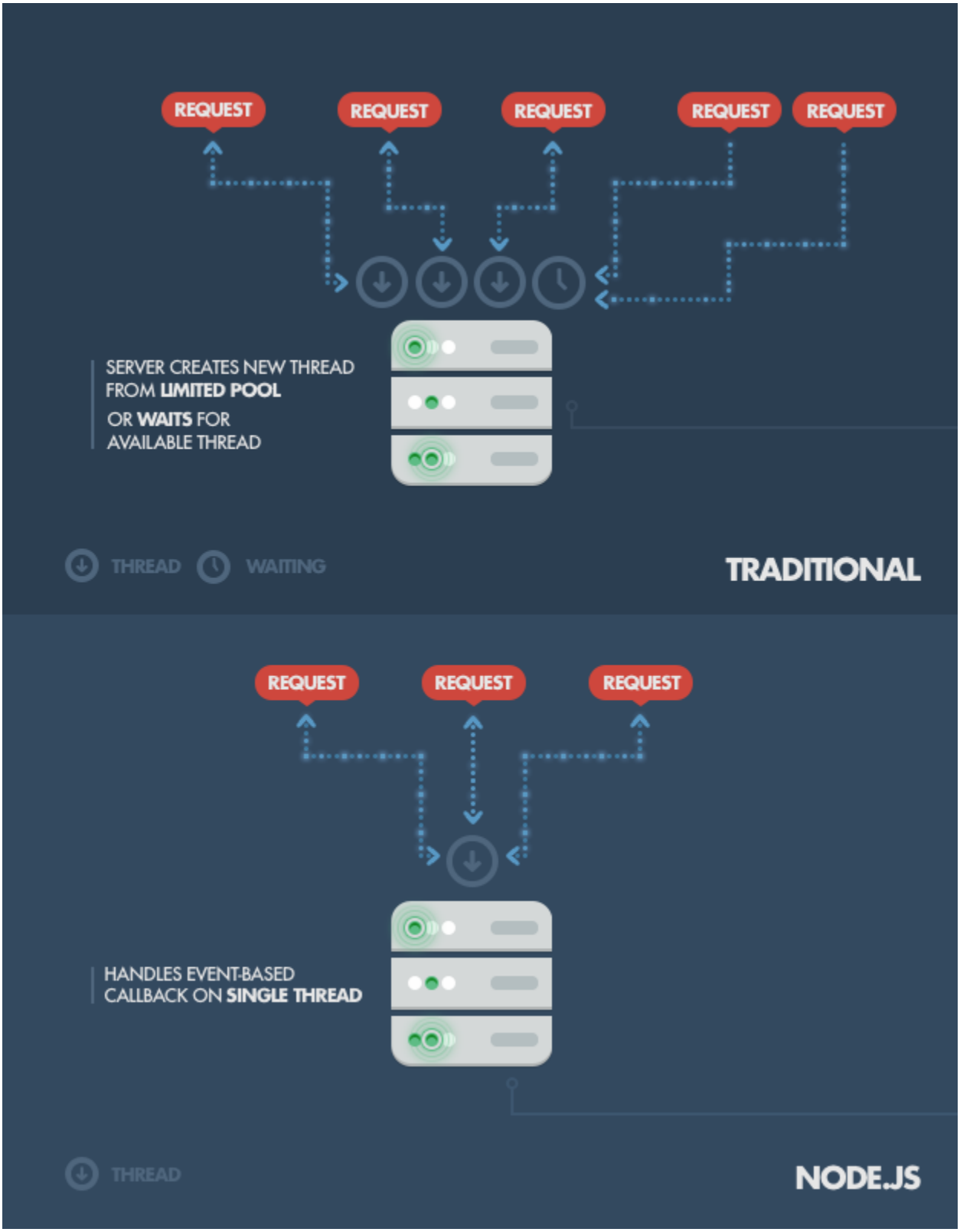
Preocupámo-nos em usar conteúdo OpenSource e/ou Standart com todas as tecnologias presentes neste trabalho.

HTML5 é um standart na web. Graças à sua escalabilidade, e funções nativas tais como <video> podemos facilmente reproduzir qualquer conteúdo sem precisar players e/ou plugins proprietários.

Node.js é uma plataforma *OpenSource* construída em JavaScript para a construção rápida de aplicações de rede, facilmente escaláveis. Node.js usa um event-driven, non-blocking I/O model (modelo baseado em eventos que permite o processamento de informação antes que a transmissão se feche) que o torna leve e eficiente, ideal para aplicações em tempo real de dados intensivos que são executados em todos os dispositivos distribuídos. WebSockets Node.js oferecem a possibilidade de "push" isto é, o cliente não tem de estar sempre a pedir informação. Sempre que o servidor recebe algo novo envia ao cliente, mantendo-o sempre atualizado.

Node.js funciona melhor em aplicações web em tempo real utilizando tecnologia "push" *websockets*. O que tem isto de tão revolucionário?

Bem, depois de mais de 20 anos de funcionamento baseado no paradigma de *request-response* sem monotorização de estado, temos finamente aplicações web com ligações em tempo real nos dois sentidos, onde o cliente e o servidor podem iniciar a comunicação, permitindo a troca de dados livremente. Isto está em contraste com o paradigma de resposta típica da web, onde o cliente inicia sempre a comunicação. Além disso, é tudo baseado em HTML, CSS e JS, sobre o porto 80.



Outros podem argumentar que já temos isso à anos sob a forma de Flash e Java Applets mas na realidade, são apenas ambientes *sandboxed* a utilizar a web como um protocolo de transporte para comunicar com o cliente. Além disso, são executados em isolamento e muitas vezes operados ao longo de portas não padrão, que podem exigir permissões extra.

Node.js é muito útil também graças à sua comunidade que produz módulos que acrescentam novas possibilidades trabalhando com todo o tipo de ferramentas (Sql, API's, etc..). Um entre muitos módulos usado neste trabalho foi o BinaryJs que permite que toda a informação que passe do servidor para o cliente e vice-versa vá em binário. A Serialização com o binaryJs é rápida e os dados permanecem em binário de um lado ao outro. Isto faz com que a transmissão de dados se torne mais rápida, uma vez que os dados já não passam em XML ou JSON como acontece nos outros tipos de transmissão por *Polling*.

BinaryJS é uma estrutura leve que utiliza websockets para enviar, fluxo e dados binários bidireccionalmente entre o browser e Node.js (backend).

Estes foram os módulos que usámos em NodeJs:

binaryjs - Binary realtime streaming

https://www.npmjs.org/package/binaryjs

express - Minimalist web framework

https://www.npmjs.org/package/express

favicon - Favicon serving middleware with caching

https://www.npmjs.org/package/static-favicon

morgan - HTTP request logger middleware

https://www.npmjs.org/package/morgan

body-parser - Body parsing middleware

https://www.npmjs.org/package/body-parser

method-override - Override HTTP verbs

https://www.npmjs.org/package/method-override

errorhandler - Development-only error handler middleware

https://www.npmjs.org/package/errorhandler

Graças a estas tecnologias conseguimos criar uma arquitectura baseada apenas em JavaScript, uma tecnologia openSource onde temos acesso a todo o código fonte, não necessitando de plugins e/ou outras linguagens. Em alternativa poderíamos ter usado PHP, C#, JAVA, C++, entre outros para o lado do backend, bem como ASP.NET/ Silverlight, JAVA Applets, FLASH, entre outros, mas utilizamos apenas javascript e HTML5, standarts na web.

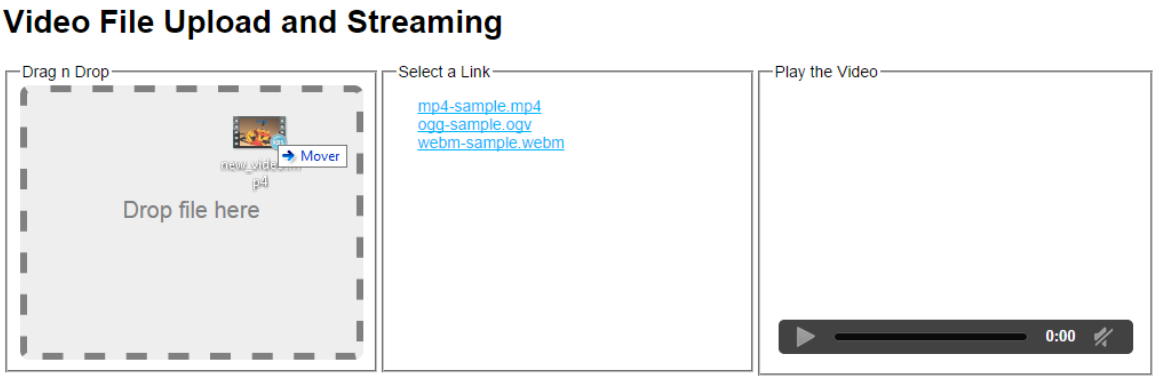
Workflow para as componentes cliente e servidor da solução:

Do lado do Servidor

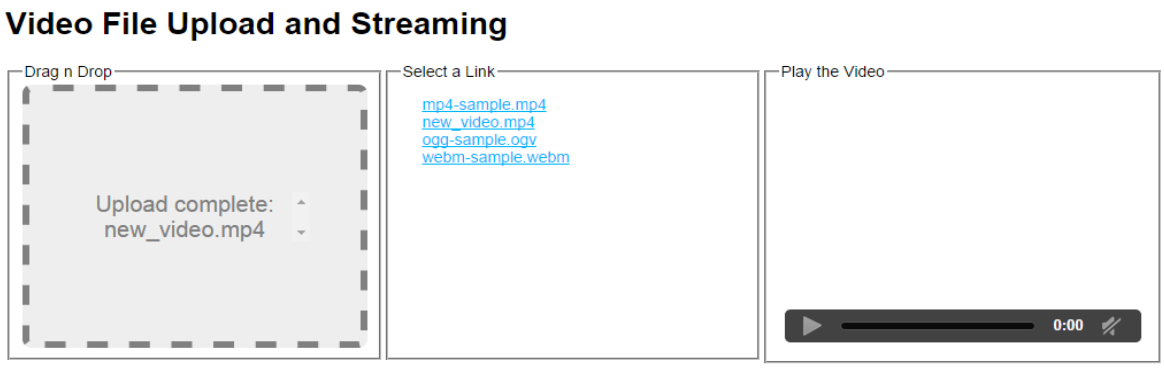
1 - Criação de uma instância do servidor BinaryJS
2 - Registar "custom events" e "handlers" para:
  - upload de videos
  - solicitar um video
  - listar videos disponíveis

Do lado do cliente

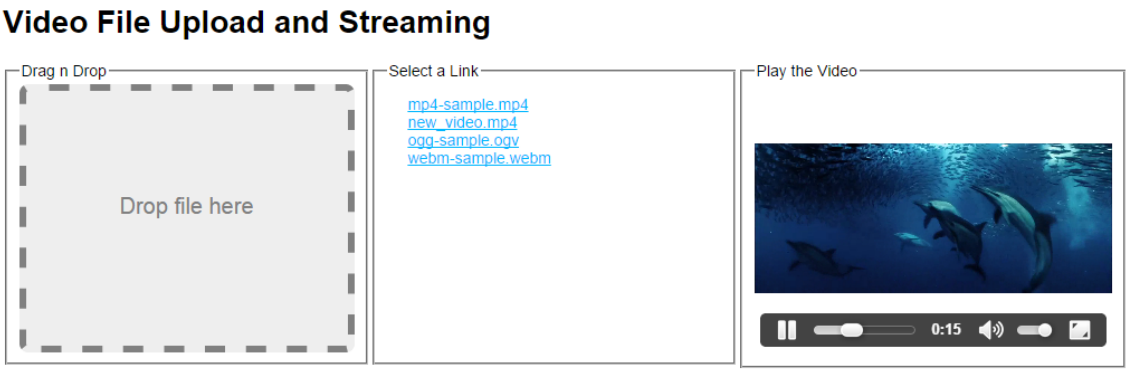
1 - Criação de uma instância do cliente BinaryJS
2 - Ao conectar-se ao servidor BinaryJS, recuperar uma lista de vídeos disponíveis e apresentá-la
3 - Ao clicar num link na lista de vídeos deve carregar o vídeo seleccionado
4 - Adicionar um meio para fazer upload de ficheiros de vídeo:
  - utilizar Drag n Drop
  - atualizar a lista de vídeos disponíveis



Upload de um vídeo para a plataforma.



Listagem do vídeo.



Reprodução do conteúdo selecionado.

COMPARAÇÃO COM OUTROS SISTEMAS

Localhost Test

A vantagem de *WebSockets* sobre o AJAX é a de que existe menos *overhead* HTTP. Depois da conexão ter sido estabelecida, todas as mensagens futuras passam pelo um *socket* e não por HTTP *request calls*. Por isso pode se assumir que o *WebSocket* pode enviar e receber mais mensagens por unidade de tempo.

O teste de comparação entre AJAX e *WebSockets* foi realizado por Peter Bengtsson.

```
# /ajaxtest (localhost)

start!

Finished

10 iterations in 0.128 seconds meaning 78.125 messages/second

start!

Finished

100 iterations in 0.335 seconds meaning 298.507 messages/second

start!

Finished

1000 iterations in 2.934 seconds meaning 340.832 messages/second


# /socktest (localhost)

Finished
```

```
10 iterations in 0.071 seconds meaning 140.845 messages/second

start!

Finished

100 iterations in 0.071 seconds meaning 1408.451 messages/second

start!

Finished

1000 iterations in 0.466 seconds meaning 2145.923 messages/second
```

Latency test

Os testes em localhost dizem que a versão WebSocket cinco vezes mais rápido que AJAX.

Obviamente que testes em localhost são irrealistas porque não têm em conta a latência e não têm em consideração as grandes distancias que os dados têm de viajar entre cliente e servidor.

Por isso os testes seguintes foram feitos com a aplicação de teste em Londres e com o servidor na Califórnia:

```
# /ajaxtest (sockshootout.peterbe.com)

start!

Finished

10 iterations in 2.241 seconds meaning 4.462 messages/second

start!

Finished

100 iterations in 28.006 seconds meaning 3.571 messages/second

start!

Finished

1000 iterations in 263.785 seconds meaning 3.791 messages/second


# /socktest (sockshootout.peterbe.com)

start!

Finished

10 iterations in 5.705 seconds meaning 1.752 messages/second

start!
```

Finished

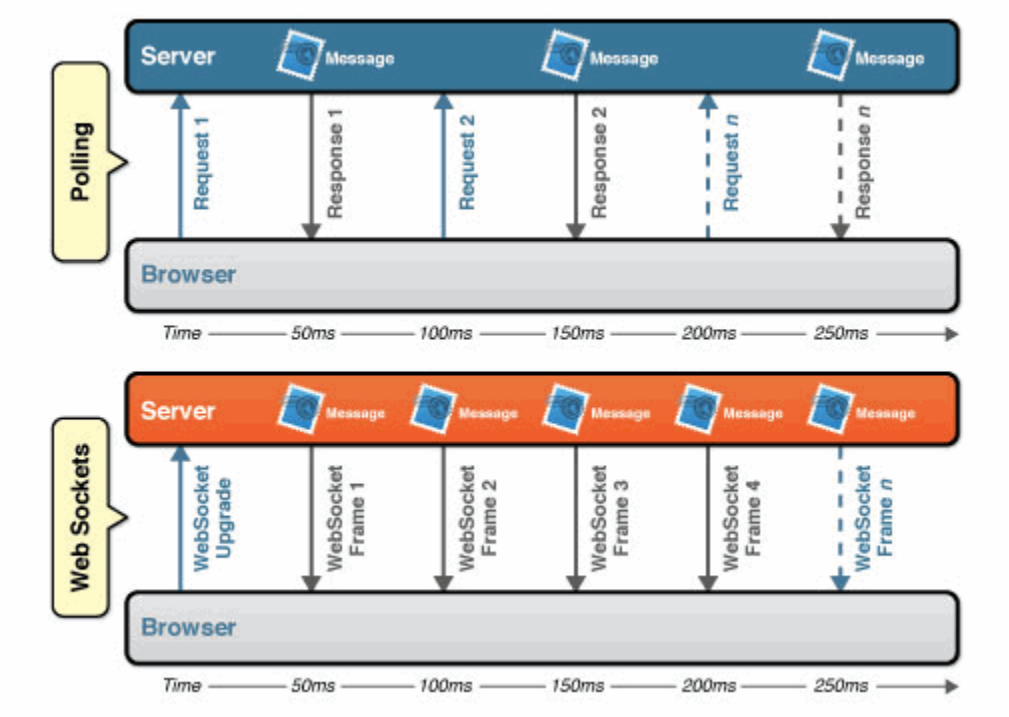
100 iterations in 23.283 seconds meaning 4.295 messages/second

start!

Finished

1000 iterations in 227.728 seconds meaning 4.391 messages/second

O teste prova que os *WebSockets* são ligeiramente mais rápidos que o AJAX entre 10-20%. Com esta diferença tão pequena os testes podem ser influenciados por vários factores como *browser* ou problemas de conecção, podemos aprender dos testes que a latência provoca perdas significativas de performance, logo converter uma aplicação já existente de AJAX para *WebSockets* não é algo estritamente necessário, visto que não teremos grandes vantagens a nível de performance.



Latency comparison between the polling and *WebSocket* application

CONCLUSÕES E MELHORAMENTOS FUTUROS

Com isto podemos concluir que os *WebSockets*, além de serem um projecto recente e de terem alguns problemas relacionados com *proxy's* e *firewalls*, têm futuro em sistemas onde a representação em tempo real de informação seja expressamente necessária, mas apenas para sistemas construídos de raiz para funcionarem com os *Websockets*, devido ao problemas apresentados acima.

Em relação aos melhoramentos futuros, os *WebSockets* podem ser usados num ambiente multiplataforma uma vez que o suporte para estes já está presente nas últimas versões do Android OS e do iOS, em todos os *browser's* modernos e até em algumas *Smart TV´s*, uma área onde os *WebSocket's* devem ser utilizados é no envio de vídeo em directo e adaptativo, já que como o vídeo adaptativo é enviado em *"chunks"* e estes têm sempre um atraso desde o pedido até à entrega, os *WebSockets* iriam eliminar este atraso já que a conecção entre cliente e servidor está sempre aberta.

REFERÊNCIAS

http://www.html5rocks.com/pt/tutorials/websockets/basics/

http://lucumr.pocoo.org/2012/9/24/websockets-101/

http://en.wikipedia.org/wiki/WebSocket

http://www.peterbe.com/plog/are-websockets-faster-than-ajax

http://www.binaryjs.com/

http://www.nodejs.org/

http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

http://pt.wikipedia.org/wiki/TCP/IP

http://en.wikipedia.org/wiki/XMLHttpRequest

http://ajuda.sapo.pt/faq.html?faq_id=69891&servico_id=7610

http://pt.wikipedia.org/wiki/REST

http://pt.wikipedia.org/wiki/Proxy

http://pt.m.wikipedia.org/wiki/Streaming

http://windows.microsoft.com/pt-pt/windows/what-is-firewall#1TC=windows-7

http://pt.wikipedia.org/wiki/Framework

GLOSSÁRIO

- **API:** É um conjunto de rotinas e padrões estabelecidos por um *software*, invocados aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar os seus serviços
- **HTTP:** Em Inglês *"Hypertext Transfer Protocol"* é um protocolo de comunicação situado na camada aplicação do Modelo OSI. Normalmente utilizado para obter páginas de Internet.
- **XMLHttpRequest (XHR):** É uma API usada para fazer pedidos http ou https para um serviço e receber resposta desse mesmo pedido.
- **RESTFUL:** é uma técnica de engenharia de software para sistemas hipermídia distribuídos como a WEB.
- **Browser:** Ferramenta que permite ao utilizador navegar e vizualisar páginas de web.
- **Proxy:** é um servidor intermediário que atende os pedidos dirigidos ao backend.
- **websocket:** É uma tecnologia que permite a comunicação bidirecional por canais full-duplex sobre um único soquete Transmission Control Protocol (TCP).
- **Firewall:** Uma firewall é software ou hardware que verifica as informações recebidas a partir da Internet ou de uma rede e que bloqueia ou permite a respectiva passagem para a máquina.
- **Backend:** Servidor responsável por tartar pedidos e responder ao cliente.
- **Polling:**
- **Framework:** É uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.
- **Downstream:** Mecanismo de streaming do Servidor para o cliente.
- **Upstream:** Mecanismo de streaming do cliente para o Servidor.
- **Streaming:** Forma de distribuição de dados, geralmente de multimedia em uma rede através de pacotes.
- **Push:** Assim que a informação contida no servidor é actualizada este comunica o cliente.
- **TCP:** É um protocolo orientado a conexões confiável que permite a entrega sem erros de um fluxo de bytes.