

Relatório do Trabalho Prático

**Docente:**  
Luís Falcão

18 de Maio de 2011

18 de Maio de 2011

18 de Maio de 2011

18 de Maio de 2011

**Discentes:**  
Diogo Fortes, 32828

João Silvestre, 32766

Ricardo Teixeira, 31737

Programação na Internet 2010/2011

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

*Instituto Superior de Engenharia de Lisboa*

Índice

[Introdução 3](#_Toc298381376)

[Cliente 4](#_Toc298381377)

[Introdução 4](#_Toc298381378)

[Padrão MVC 4](#_Toc298381379)

[Servidor 5](#_Toc298381380)

[Introdução 5](#_Toc298381381)

[Controllers 5](#_Toc298381382)

[Views 6](#_Toc298381383)

[Módulo de Autenticação 7](#_Toc298381384)

[Partes Individuais 8](#_Toc298381385)

[Introdução 8](#_Toc298381386)

[Grupo de Amigos 8](#_Toc298381387)

[Implementação 8](#_Toc298381388)

[Chat 8](#_Toc298381389)

[Protocolo 8](#_Toc298381390)

[Implementação Servidor 9](#_Toc298381391)

[Implementação Cliente 10](#_Toc298381392)

[Conclusão 12](#_Toc298381393)

# Introdução

# Cliente

### Introdução

A implementação do lado cliente foi feita através da utilização de HTML/XHTML, DOM (Document Object Model), JavaScript, Ajax e CSS (Cascading Style Sheets). Foi utilizado HTML e CSS para criar as páginas WEB dando o aspecto pretendido a aplicação WEB, DOM e JavaScript para a criação, remoção, modificação e navegação nos elementos de um documento HTML/XHTML, bem com como para a criação de scripts (JavaScript) para uma resposta a acções por parte do utilizador mais rápidas. Por último foi utilizada a arquitectura AJAX para o pedido assíncrono de dados ao servidor. De seguida iremos explicar os componentes criados e a sua utilização.

### Padrão MVC

De modo a separar a lógica da aplicação da apresentação, a implementação do cliente foi feita utilizando o padrão MVC (Model-View-Controller). Assim foram criados os ficheiros:

* Model.js
* View.js
* Controller.js

O ficheiro Model encarrega-se de gerir do domínio da aplicação e notificar quem estiver a observar sobre as mudanças nos dados. Neste caso o Model é o responsável por gerir a criação e remoção de shares (dos tipos existentes) de um utilizador, passando essa informação ao servidor de modo a ser tornada persistente. Esse envio de informação é feito de forma assíncrona através da arquitectura AJAX.

O ficheiro View é o responsável pela apresentação dos dados ao utilizador. Assim este ficheiro é responsável por apresentar os shares que vão sendo criados pelo utilizador, ou, então, remover visualmente o share.

O ficheiro Controller é a cola entre os dois ficheiros anteriores sendo o responsável por receber as notificações do utilizador e trata-las da maneira mais corrente, invocando objectos do modelo e de seguida apresentando-os através dos métodos contidos no View.

Por último foi criado também um ficheiro de nome Wall.js que faz a actualização periódica????

# Servidor

### Introdução

O servidor web foi implementado utilizando a framework MVC 3, que funciona sobre ASP.NET. Esta framework funciona à base de views e controllers sendo que os controllers funcionam como o processamento de um pedido http e a view representa a página web a enviar ao utilizador. Nesta secção irão ser explicados os componentes, destes tipos, e qual a sua utilização.

### Controllers

Para implementar todas as funcionalidades pedidas no enunciado foram criados 4 controllers:

* AccountController
* HomeController
* SharesController

#### AccountController

O *AccountController* é responsável por todas as acções relacionadas com as contas do utilizador, ou seja, é aqui que se encontram as acções de entrada (log in), saída (log out) e registo na aplicação.

A acção de ***log in*** é responsável por verificar as credenciais do utilizador e caso estas sejam válidas irá criar um cookie, utilizando um método do módulo de autenticação que sabe criar um cookie válido, de forma a marcar o utilizador como alguém que já entrou.

A acção de ***log out*** é responsável por eliminar o cookie do utilizador para que em futuros pedidos este já não seja considerado como alguém que entrou, para isto é usado também um método do módulo de autenticação.

A acção de **registo** é responsável por verificar se o *username* se encontra disponível, se não tiver deve retornar a mesma página com uma mensagem de erro a identificar o problema, caso este se encontre disponível deve então registar o utilizador e fazer *log in* desse utilizador. Este também é responsável pela visualização e alteração de um perfil de utilizador.

#### HomeController

O *HomeController* é responsável pela visualização da página inicial da aplicação e pela Wall do utilizador.

#### SharesController

O *SharesController* é responsável pela criação, eliminação e obtenção de shares. A acção de **criar** shares é responsável por associar um novo share ao utilizador, este recebe o tipo do share e usando reflexão irá criar um novo share desse tipo, portanto para adicionar um novo share basta criá-lo no mesmo *namespace* do Share e estender deste.

A acção de **eliminar** trata de eliminar um share identificado por um identificador e que se encontra associado a um utilizador. A acção de **obter** retorna um conjunto de shares e dependendo dos parâmetros variam os shares. Se for apenas enviado o utilizador este retorna todos os shares de um utilizador, se for enviado o identificador do share mais recente a acção retorna os todos os novos, e se para além de esse também enviar o do mais antigo este envia todos os novos e os que devem ser eliminados.

### Views

# Módulo de Autenticação

# Partes Individuais

## Introdução

Nesta secção vão ser explicadas as partes individuais de cada membro do grupo.

* Grupo de Amigos – Diogo Fortes
* Chat – João Silvestre
* Autênticação através do Facebook – Ricardo Teixeira

## Grupo de Amigos

O grupo de amigos tem como finalidade a adição, a uma lista, dos utilizadores conhecidos de modo a que estes sejam de fácil acesso por parte do utilizador pois são utilizadores que conhece real ou virtualmente e com quem mantém um contacto regular.

### Implementação

Para a implementação desta funcionalidade foi criado um Mapper que trata das adições e remoções de utilizadores à lista de amigos de um dado utilizador chamado de “FriendsMapper.cs”. De modo a tratar a concorrência de pedidos por parte dos utilizadores esta classe utiliza mecanismos de sincronismo para as adições e remoções à lista.

Ao nível da apresentação da lista de amigos foi criada a View Friends.cshtml que recebe um utilizador e apresenta o seu grupo de amigos. Nesta apresentação é adicionado um botão de Eliminar, por cada amigo, para que o utilizador possa remover um utilizador da lista de amigos. Para além disto foi também adicionado à View ViewProfile.cshtml um troço de código para que quando estivesse a ser mostrado o perfil de um utilizador que não o autenticado, fosse adicionado um botão com o nome “adicionar como amigo” para que o utilizador autenticado possa adicionar outros utilizadores à sua lista de amigos.

Por último, foram acrescentados ao Controller AccountController.cs os métodos AddFriend e RemoveFriend que são invocados pelas Views e controlam as acções de adição e remoção de utilizador através dos Mappers UserMapper.cs e FriendsMapper.cs descritos em cima.

## Chat

O chat na aplicação funciona como ponto de encontro entre todos os utilizadores que lá entrem. Este foi desenvolvido recorrendo à tecnologia WebSockets do HTML5, que permite a criação de um socket permitindo que tanto o cliente como o servidor possam usar o socket para comunicar. Utiliza-se um servidor de WebSockets chamado SuperWebSockets no desenvolvimento desta funcionalidade.

### Protocolo

No desenvolvimento do chat foi necessário estebelecer um protocolo de comunicação para identificar os comandos que tanto o cliente como o servidor poderiam executar, as acções que o protocolo suporta são representadas por apenas um carácter no inicio da mensagem, como forma de reduzir o tráfego. Este protocolo permite os seguintes comandos:

* Enviar uma lista dos utilizadores online (Servidor -> Cliente), representado pelo carácter L;
* Anunciar a entrada de utilizador (Servidor -> Cliente), representado pelo carácter U;
* Anunciar a saída de um utilizador (Servidor -> Cliente), representado pelo carácter R;
* Enviar mensagens (Servidor <-> Cliente), representado pelo carácter M;
* Enviar mensagens privadas (Servidor <-> Cliente), representado pelo carácter P.
* Erro (Servidor <-> Cliente), representado pelo carácter E).

### Implementação Servidor

O desenvolvimento desta funcionalidade implicou então a criação da componente servidora e da componente cliente. A componente servidora (CharServer.cs), utiliza como dito anteriormente a implementação SuperWebSockets, e utilizando esta implementação são apenas precisos 3 passos. Esta implementação permite a configuração do servidor através do ficheiro de configuração e utilizando uma versão normal dos sockets a configuração é a seguinte:

<socketServer>

<servers>

<server name="SuperWebSocket"

serviceName="SuperWebSocket"

ip="Any" port="4500" mode="Sync">

</server>

</servers>

<services>

<service name="SuperWebSocket"

type="SuperWebSocket.WebSocketServer, SuperWebSocket" />

</services>

</socketServer>

E utilizando as seguintes linhas de código consegue-se configurar e começar o servidor:

var serverConfig = ConfigurationManager.GetSection("socketServer") as SocketServiceConfig;

SocketServerManager.Initialize(serverConfig);

SocketServerManager.Start();

E apenas com estas 3 linhas de código tem-se o servidor a correr, no entanto, é preciso agora conseguir interagir com o servidor. Para esse efeito eles permitem registar, a quando da inicialização do servidor, 3 delegates para as seguintes acções:

* Novo socket aberto;
* Socket fechado;
* Mensagem no socket.

Vamos então ver como foram usados estes 3 delegates para a implementação do chat. Quando se abre um novo socket significa que existe um novo utilizador, sendo assim são feitas 3 coisas. A primeira é avisar todos os utilizadores ligados que existe um novo utilizador, utilizando o comando U podemos então alertar os outros para esse efeito. É preciso então registar o representante do socket deste utilizador para mais tarde ser usado para enviar comandos, e por fim é preciso enviar-lhe uma lista dos utilizadores ligados para que este possa mostrar quem se encontra online.

Quando o socket é fechado é então preciso notificar todos os outros utilizadores que o utilizador X se desligou e é preciso actualizar a colecção dos sockets para garantir que esta representa a realidade.

E por fim o delegate de quando se recebe mensagens, só existem dois tipos de mensagens neste chat as mensagens globais e as mensagens privadas. As mensagens privadas começam por @ seguido do utilizador para qual esta é destinada, terminando com um espaço. É portanto verificado se a mensagem começa por @, se não começar basta apenas enviar a mensagem para todos os outros sockets para que todos os utilizadores ligados a recebam. Caso come-se pelo @ é então preciso extrair o utilizador à qual é destinada a mensagem, ver se este se encontra na lista dos utilizadores ligados e caso exista enviar a mensagem para o utilizador (já sem o @<user>). Caso não exista é preciso enviar uma mensagem de erro para o utilizador que tentou enviar a mensagem.

Registando então estes 3 delegates e iniciando o servidor temos um servidor de chat pronto a ser usado, é então preciso criar a aplicação cliente.

### Implementação Cliente

O cliente irá então usar WebSockets para comunicar com o servidor e para o fazer basta apenas, em javascript, criar um objecto do tipo WebSockets e dar-lhe o URL do servidor (Nota que de momento apenas browsers WebKit suportam WebSockets, ou seja, chrome e safari).

Este objecto permite então registar funções para vários tipos de callbacks que possam acontecer, entre eles encontram-se:

* Onerror, chamado quando existe um erro a estabelecer a ligação com o servidor;
* Onopen, chamado quando a ligação é estabelecida;
* Onclose, chamado quando a ligação é fechada;
* Onmessage, chamado quando é enviada uma mensagem do servidor para o cliente.

Os 3 primeiros são apenas usados, no nosso cliente, para efeitos informativos o evento que realmente interessa é o “onmessage”. A função registada neste evento tem apenas uma função, identificar o tipo do comando e enviar para a função correcta para o processar.

Foi feita uma abordagem ao problema usando o padrão MVC sendo que o modelo é o responsável pela comunicação com o servidor, as funções de processamento dos comandos são dadas pelo controller que regista as funções no modelo, este é então responsável por dar informação à view para esta mostrar ao utilizador.

# Conclusão