INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Engenharia Informática e de Computadores

Programação Internet

MinesweeperFlags

Membros do Grupo					
32223	Paulo Pires	31923	Nuno Sousa	26657	Ricardo Neto

Indice

ServerSide

Implementação Individual (Aluno #32223)

Implementação Individual (Aluno #31923)

Implementação Individual (Aluno #26657)

1 Introdução

Pretende-se com este trabalho, não só dar resposta ao desafio proposto, o da implementação do jogo MineSweeperFlags, mas também o de demonstrar os conhecimentos adquiridos ao longo do semestre por parte dos elementos deste grupo.

De acordo com os requisitos, a solução final deve ser uma junção homogénea das resoluções dos diferentes enunciados propostos ao longo do semestre, requisito esse que foi cumprido. Todavia, foram tomadas decisões em cada fase, com vista a atingir os objectivos propostos, que, naturalmente, comprometeram as implementações das fases seguintes. Era de esperar que tal acontecesse e, apesar da solução final não estar independente dessas decisões, o grupo tem noção de quais os aspectos que podiam ter sido melhorados.

2 Descrição Funcional

A abordagem foi a de criar uma solução em que um utilizador, após registo e/ou Login, tenha acesso a um espaço único, denominado Lobby, que coloca ao seu dispor diversas funcionalidades, entre as quais:

- Visualização de jogadores online
- Visualização de jogos públicos criados
- Chat room com possibilidade de envio de mensagens públicas ou privadas
- Gestão de lista de amigos
- Criação de jogos públicos ou privados
- Edição de perfil

A estrutura do Lobby foi pensada por forma a permitir que um jogador possa estar no chat e, em simultâneo, possa estar a jogar um ou mais jogos. Para tal criou-se uma barra de separadores que irá conter os jogos em que o jogador participa.

2.1.1 Registo de Utilizador

O primeiro passo para um novo utilizador poder aceder ao Lobby é registar-se. Para tal, é disponibilizado um link na página de entrada (Game/Start) que irá direccionar o utilizador para a página de registo (Profile/Create).

Fig. 1 - Página de Entrada

Aqui deverá ser indicado o endereço de e-mail, nome, password, indicar se está online ou offline e o caminho para o ficheiro que será a sua imagem.

Fig. 2 - Página de Registo

2.1.2 Jogadores, jogos, mensagens e amigos

Após fazer login/registo o utilizador é encaminhado para a página principal do Lobby (Game/Main), estando a mesma descrita com maior pormenor mais à frente, no que respeita à sua estrutura.

Fig. 3 - Página Principal do Lobby

Do lado esquerdo está a informação do jogador logado, dos seus amigos e as acções de criação de jogo e edição de perfil. Ao centro está a área de comunicações onde o jogador ve as mensagens públicas trocadas entre os diversos jogadores, as mensagens privadas enviadas para ele, envia mensagens públicas ou privadas e a área onde estarão disponiveis os convites feitos por outros jogadores. No lado direito estão listados os jogadores logados e os jogos disponíveis.

2.1.3 Criação de Jogos

O jogador pode optar, como já foi dito, pela criação de jogos públicos ou privados. Em qualquer um dos casos é necessário indicar o nome de um jogo que ainda não esteja em uso.
A criação de um jogo irá criar um separador com o nome do jogo, sendo que, ao ser seleccionado, irá ser gerada a página de jogo onde o owner podeá dar inicio assim que estiver presente pelo menos mais um jogador. No caso de um jogo privado, só se pode proceder a sua criação quando está seleccionado pelo menos um amigo e menos de 3, para quem será enviado um convite.

2.1.4 Envio de Mensagens

O envio de mensagens procede-se de forma simples, bastando ao utilizador introduzir o texto no controlo disponibilizado para o efeito e carregar no botão de envio para todos ou para um utilizador em particular, sendo que, no último caso, esse jogador deve ser seleccionado na combo box disponibilizada. As mensagens são mostradas na caixa de mensagens em cima.

2.1.5 Jogadores online e amigos

Na área de jogadores online vão aparecendo os jogadores que se logam na aplicação. É possível escolher um jogador e adiciona-lo à lista de amigos. Após estar presente na lista de amigos, é possível remove-lo e, conforme mencionado anteriormente, convida-lo para um jogo privado. Um jogador que esteja logado aparece na combo box de jogadores para onde podem ser enviadas mensagens privadas.

2.1.6 Jogos disponíveis

Na área de jogos disponíveis estão listados os jogos que foram criados, havendo a possibilidade do jogador de juntar a um deles, na qualidade de participante.

2.1.7 Recepção e envio de convites

A recepção de convites acontece quando somos seleccionados no âmbito da criação de um jogo privado. O convite permite aceitação, que irá criar um separador com o jogo, ou recusa, que enviará uma mensagem privada ao jogador que criou o jogo.

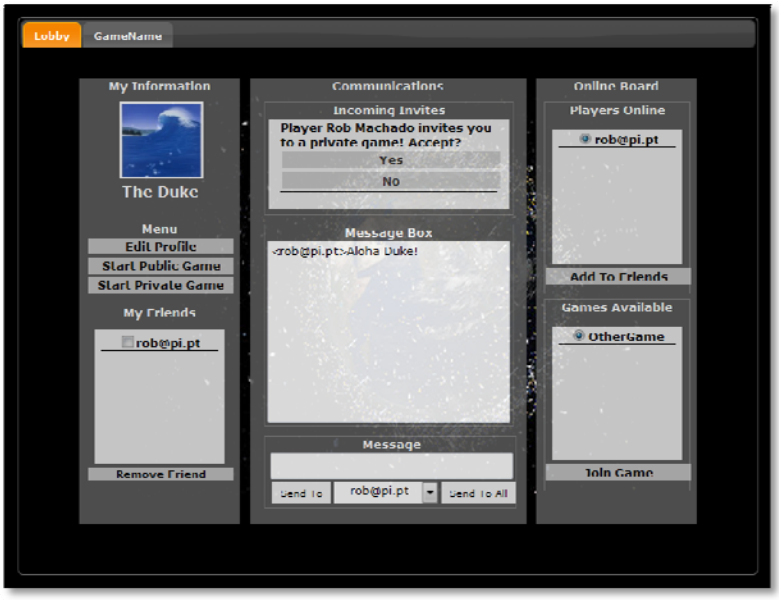


Fig. 4 - Lobby com jogador online, amigo, jogo disponivel, mensagem e convite

2.1.8 Jogo

Como se pode ver na figura 4, Está disponível, na barra de separadores, um jogo com o nome "GameName". Ao clicar nesse separador, é feito um pedido ao servidor de uma página com um novo jogo ou, um jogo já existente com o nome indicado.

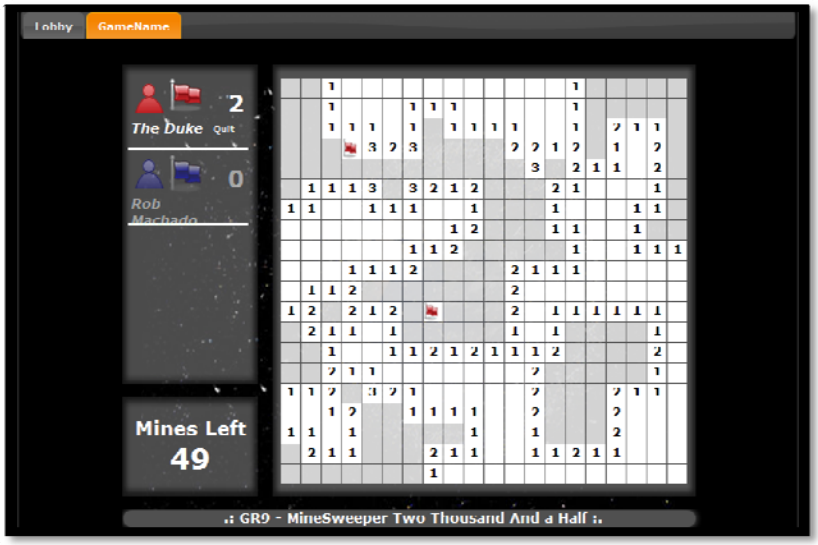


Fig. 5 - Ecran de jogo

O tabuleiro de jogo apresenta-se no formato 20x20. Por omissão contém 51 minas, contudo, este número pode diminuir para 50, no caso de um jogo de 3 jogadores, uma vez que 51 é múltiplo de 3, logo permitira que acontecesse situações de empate. Ao longo do jogo também é possível que o número de minas venha a diminuír sem que tenham sido encontradas para evitar situações de empate (e.g. no caso de um jogo de 3 jogadores, se existirem 42 minas no tabuleiro e um dos jogadores desistir o número de minas é ajustado para 41). Ganha o primeiro jogador que tiver encontrado a maioria das minas. Cada jogador pode optar por desistir a qualquer momento. Ao terminar um jogo é possível visualizar a posição das restantes minas. De notar que o facto do jogador vermelho ter ganho não foi coincidência.

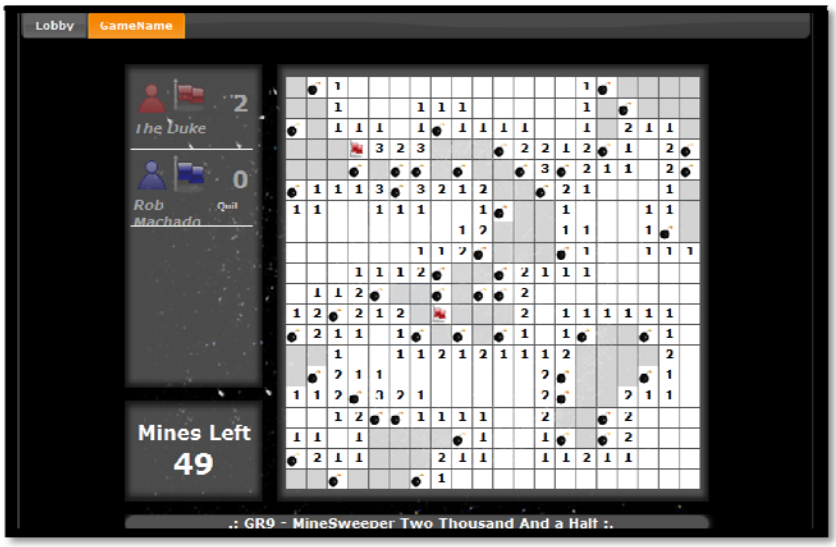


Fig. 6 - Final do jogo com visualização das minas por encontrar

3 Client Side

3.1 BoxModels

Apesar do elemento div que corresponde à funcionalidade de separadores estar apenas presente na página que representa Lobby, devido ao tipo de funcionalidade que o jQuery proporciona, pode-se dizer que tanto Lobby como Game estão contidos em Tabs.

3.1.1 Lobby

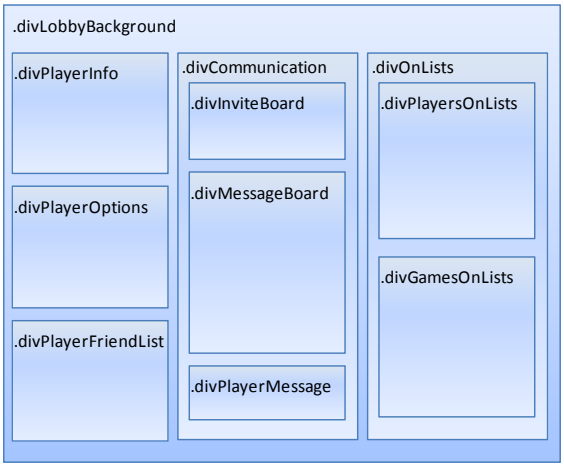


Fig. 7 - BoxModel de Lobby

3.1.2 Game

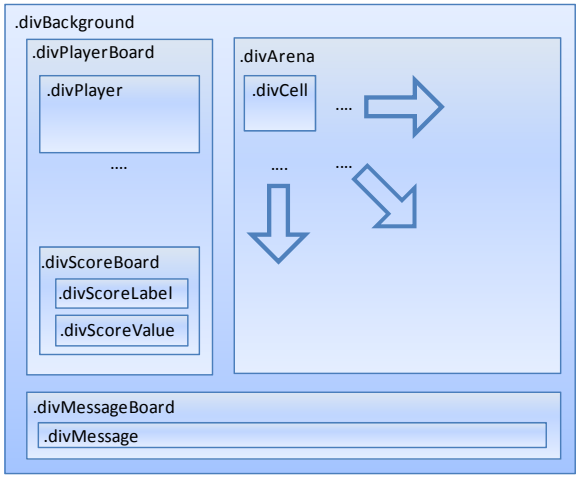


Fig. 8 - BoxModel de Game

A estrutura de cada Jogador é composta por vários elementos, nomeadamente imagem, nome, pontuação e botão de quit. Esta estrutura não está presente na página de game (Views/Game/Show.aspx) uma vez que é gerada dinamicamente em JavaScript.

3.1.2.1 PlayerBoard

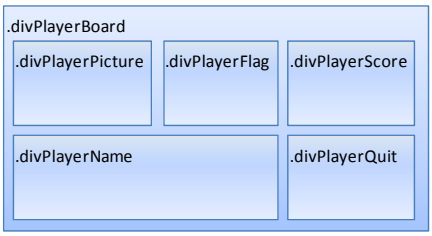


Fig. 9 - BoxModel de PlayerBoard

3.2 UML Client Side

O código da aplicação que corre do lado do cliente obedece ao seguinte diagrama:

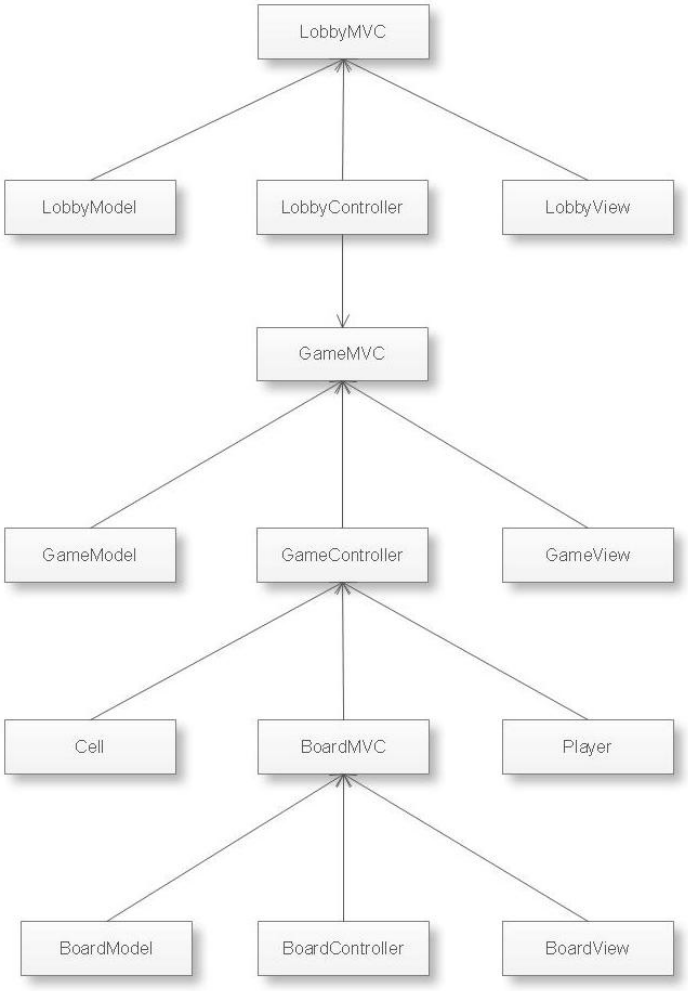


Fig. 10 - UML de JavaScripts

3.2.1 LobbyMVC

LobbyMVC é o responsável pela gestão da página de início da aplicação, fazendo para isso uso das respectivas funções LobbyController, LobbyView e LobbyModel.
LobbyView é responsável por manusear os elementos presentes no HTML a pedido de LobbyController. É em LobbyView que são criados os separadores e, é neste aspecto que não é cumprido o padrão MVC, na medida em que, é LobbyView que determina quais os endpoints responsáveis por criar um jogo e por permitir que se participe num jogo já existente.

LobbyModel guarda informações relativas ao jogador, nomeadamente, nome e Email. Este último é o que permite ao Cliente identificar-se perante o servidor, uma vez que, foi determinado que o Email serviria de "chave primária".

LobbyController é responsável por interagir com o servidor, bem como, agilizar a ligação entre LobbyView e LobbyModel, de acordo com o padrão MVC. É em LobbyController que estão definidos os mecanismos de pooling necessários à obtenção de informações do servidor de forma assíncrona. Todo o handling de eventos é também feito em LobbyController. Quando é iniciado um novo jogo ou quando se inicia a participação em um jogo existente, é instanciado um novo objecto GameMVC que manipulará elementos num separador específico.

3.2.2 GameMVC

GameMVC é o responsável pela gestão do jogo, utilizando o objecto Player, BoardMVC e Cell.
GameView é responsável por manusear os elementos presentes no HTML.

GameModel guarda informações relativas ao jogo, nomeadamente o estado do jogo, nome do jogo, nome do jogador, Email, identificador do jogador (0..3), identificador do jogador que está a jogar, flag indicadora de ser um jogo criado pelo próprio jogador ou não e número de jogadores. O estado do jogo, bem como, as constantes utilizadas, está descrito em Constants.js.

GameController é responsável por interagir com o servidor, bem como, agilizar a ligação entre GameView e GameModel. É em GameController que estão definidos os mecanismos de pooling, bem como, os handles para eventos.

3.2.3 BoardMVC e Cell

BoardMVC é um objecto que é iniciado com um número de linhas, colunas, uma "referência" para Cell e a chave única identificadora do jogo a que pertence. BoardView é responsável por desenhar todo o tabuleiro, bem como, criar as células que o constituirão.
A criação das células é feita com base na "referência" para Cell que é recebida, sendo esta referência a responsável por fazer o bind dos eventos às células. Além destas funções Cell também altera o aspecto visual de uma célula, bem como, disponibiliza informações acerca do seu tipo (mina, número, vazia).

3.2.4 Player

Este tipo é o responsável por manusear jogadores durante o jogo. Oferece funcionalidades que permitem alteração da pontuação, desenho, remoção e activação (quando é a sua vez de jogar).

3.3 Interacção com o Servidor

Do lado do cliente, o envio de pedidos ao servidor é efectuado com recurso a uma instância de *HttpRequest*, que abstrai os tipos que o utilizam de toda a lógica de construção do *XMLHttpRequest* e do *url* a solicitar e eventuais parametros (QueryString.

```
function HttpRequest(handlerClass, handlerUrl, gName, playerId) {

var xhr;
var _args = arguments;

this.Request = function() {
    xhr = new XMLHttpRequest();

    var data = "";

    for (var i = 4; i < _args.length; i += 2) {
        data += "&" + _args[i] + "=" + escape(_args[i + 1]);
    }
    var gChannel = "/" + handlerClass + "/" + handlerUrl
                + "?gName=" + escape(gName)
                + "&playerId=" + escape(playerId)
                + data;
    xhr.open("GET", gChannel, false);
    xhr.send();
    if (xhr.status != 200) throw (xhr.responseText);
}
...
```

A iniciação de HttpRequest obriga a indicação de qual o handler e url do endpoint, bem como, o nome do jogo e o identificador do jogador. As respostas são obtidas através das propriedades getJSonObject ou getresponseText.

Por forma manter os dados actualizados no vários utilizadores foi implementada uma estratégia de pooling que, à frequência de 1 Hz, questiona o servidor sobre qualque alteração de dados partilhados com outros utilizadores, assegurando desta forma o correcto funcionamento em rede e com múltiplos utilizadores.

4 ServerSide

4.1 Constituintes

A aplicação alojada no servidor é constituida por 4 projectos distintos, a saber:

- *MineSweeper*, representante do *Model* subjacente à aplicação;

- *MineSweeperControllers*, que representa os vários *Controllers* associados à aplicação;

- *MineSweeperSecurity*, que contém a lógica associada ao processo de Registo/*Login*;

- *MineSweeperGUI*, responsável pelo código a ser enviado ao cliente, nomeadamente *View*'s, *CSS*'s e *javaScripts*.

4.2 Model

O Model associado à aplicação, representado pelo projecto MineSweeper, traduz-se no seguinte diagrama:

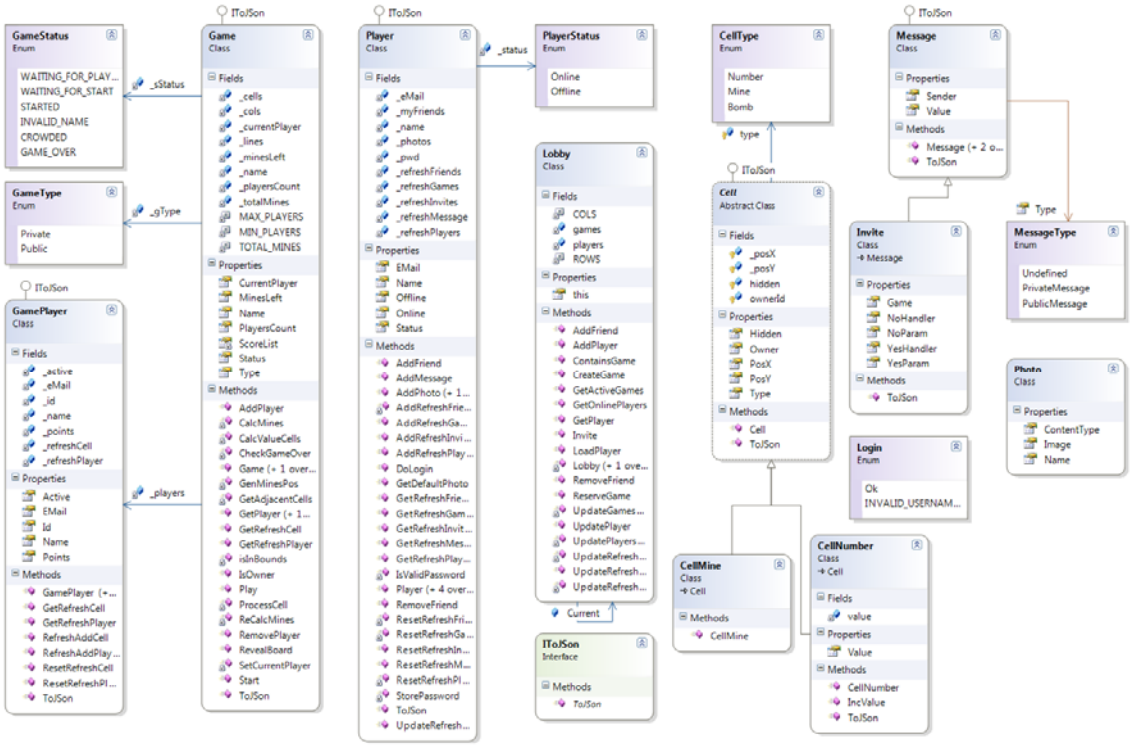


Fig. 11 - Diagrama de Classes

A classe *Lobby* contém um *SINGLETON* para a instância corrente da aplicação exposto através da propriedade *Current* iniciada no seu construtor de tipo. Uma instância de *Lobby* é composta pelas Colecções de *Game*s e *Player*s activos no momento. É ela a responsável pela criação de

novos *Game*s, pela criação e/ou carregamento de *Player*s, bem como de todas as funcionalidades inerentes ao ecrán de *Lobby*, nomeadamente, a gestão da informação a ser enviada ás estruturas de dados subjacentes ao mecanismo de pooling implementado, através do seus métodos *UpdatePlayersOnNewPlayer*, *UpdateRefreshPlayers*, *UpdateGamesOnNewPlayer*, *UpdateRefreshGames* e *UpdateRefreshMessages*, a criação de convites e o respectivo registo nos jogadores envolvidos e a adição e remoção de *Friends* de um dado Jogador.

5 WebStress Tool (Paulo Pires)

Nota introdução

Uma vez que os conhecimentos adquiridos sobre o desenvolvimento de aplicações web, recorrendo à arquitectura MVC já tinham sido consolidados e demonstrados na parte III do trabalho resolvi fazer algo diferente.
Antes de decidir o que implementar defini as áreas de domínio do software. Neste, deveria estar reflectido os conhecimentos adquiridos sobre o protocolo HTTP, ASP.NET Pipeline e de uma forma mais lata programação assíncrona.

Assim sendo, desenvolvi o software "Web Stres Tool", que não é mais do que um utilitário que permite colectar e reprocessar os pedidos dos diversos utilizadores.

Desenho técnico

O software "Web Stres Tool" é composto por dois módulos, a ver:

1. **Web Stress Tool Collector**, responsável por colectar e persistir os pedidos realizados pelos diversos utilizadores.
2. **Web Stress Tool**, utilitário que possibilita reprocessar os pedidos colectados pelo Stress Tool Collector.

Web Stress Tool Collector

Tecnicamente o Web Stress Tool Collector é um System.Web.IHttpModule que intercepta os pedidos efectuados pelos diversos utilizadores através da subscrição do evento BeginRequest do tipo HttpApplication.

A persistência do pedido é realizada para ficheiro. Neste, está representado o pedido http, no formato "raw request", que o utilizador efectuou.

A activação do Web Stress Tool Collecter é conseguida através da configuração do elemento httpModules do espaço de nomes system.web do ficherio Web.config.

```
<system.web>
    <compilation debug="true">...</compilation>
    <authentication mode="Forms">...</authentication>
    <membership>...</membership>
    <profile>...</profile>
    <roleManager enabled="false">...</roleManager>
    <pages pageParserFilterType="System.Web.Mvc." pageBaseType="System.Web.Mvc." userControlBas
    <httpHandlers>...</httpHandlers>
    <httpModules>
        <add name="ScriptModule" type="System.Web.Handlers.ScriptModule, System.Web.Extensions,
        <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, System.Web.Rout
        <add name="Security" type="MinesweeperSecurity.Security, MinesweeperSecurity" />
        <add name="Collector" type="StressToolCollector.Collector, StressToolCollector" />
```

Exemplo de um ficheiro persistido

```
GET /favicon.ico HTTP/1.1
Connection: keep-alive
Accept: */*
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,pt-PT;q=0.8,pt;q=0.6,en;q=0.4
Cookie: uu=57de034dadf83600f9c8d9831d7451ef23d2a981
Host: localhost:49505
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/532.5 (KHTML, like Gecko)
```

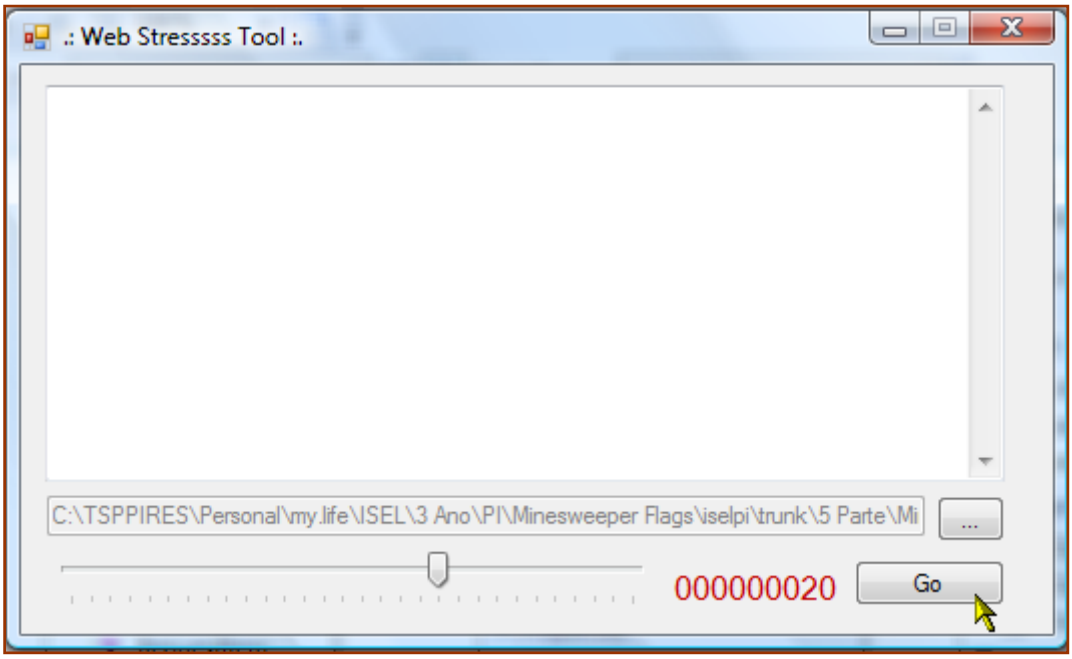
Detalhes da implementação

```
using System.Threading;
using System.IO;

namespace StressToolCollector
{
    public class Collector : IHttpModule
    {
        public Collector() { }

        string GetFilePath( Uri requestUrl )...

        void BeginRequest(object sender, EventArgs e)
        {
            if (sender == null) throw new ArgumentNullException("sender");
            HttpApplication ctx = (HttpApplication)sender;

            ctx.Request.SaveAs(GetFilePath( ctx.Request.Url ), true);
        }

        public void Init(HttpApplication ctx)
        {
            ctx.BeginRequest += new EventHandler(BeginRequest);
        }

        public void Dispose() { /* do nothing */ }
    }
```

Web Stress Tool

O utilitário Web Stress Tool realiza o reprocessamento dos pedidos colectados pelo Web Stress Tool Collector. Uma vez que estes estão no formato “raw request” não foi possível recorrer aos objectos HTTPWebRequest ou WebCliente constantes na .net framework para efectuar o reprocessamento do pedido.

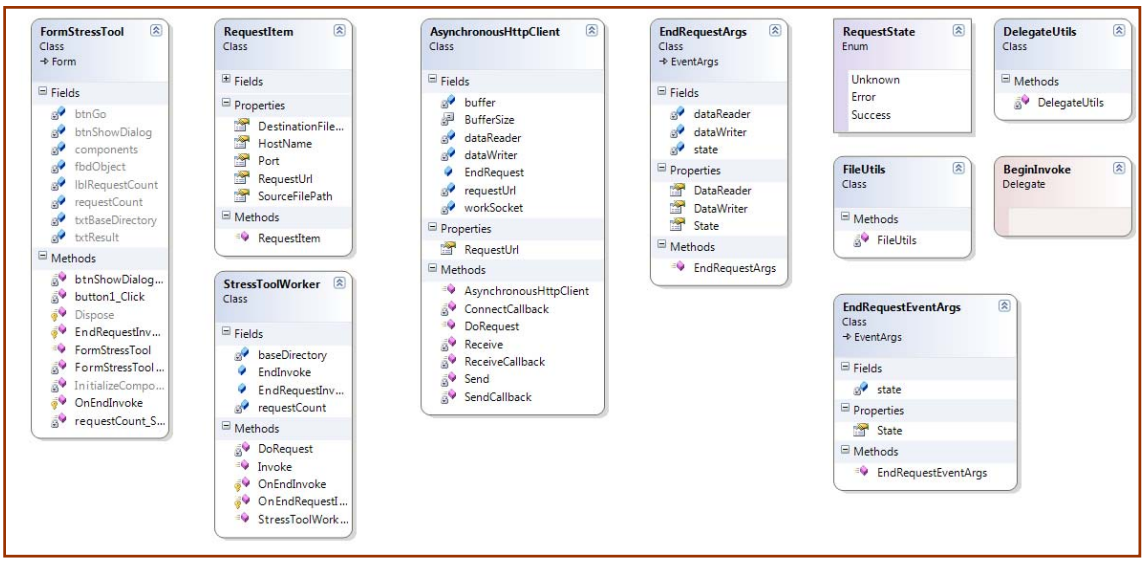
Neste sentido resolvi implementar o reprocessamento com a utilização do objecto Socket. O que transformou a implementação mais trabalhosa e delicada.



Depois de iniciada a aplicação, o utilizador deverá indicar a localização dos ficheiros dos com os pedidos. Após ter escolhido a directoria, com os respectivos ficheiros, deverá configurar o nº de vezes que quer reprocessar cada pedido. Após terminada a configuração será disponibilizado a opção Go.

Depois de seleccionada a opção Go são reprocessados os pedidos. Por cada pedido reprocessado é apresentado no ecrã a mensagem de estado associada ao pedido. (Success ou Error)

Diagrama de classes



FormStressTool, form responsável por recolher as configurações definidas pelo utilizador e iniciar o reprocessamento dos pedidos. Para o reprocessamento é utilizado o tipo *StressToolWorker* onde é passado a directoria onde estão armazenados os ficheiros com os pedidos.

O método Invoke dá inicio ao reprocessamento, para isso faz recurso do tipo *AsynchronousHttpClient*, que implementa a ligação física.

Limites da solução

- Os objectos baseados em data de expiração não são suportados funcionalmente.
- Em função da implementação especifica do código servidor, a ferramenta pode apresentar problemas na gestão do estado sessão e aplicação.
- A colecta de pedidos no ambiente de desenvolvimento e processamento em ambiente de testes, pode produzir um comportamento errado, uma vez que na encriptação do viewstate é usado o MAC Address da placa de rede. A resolução deste problema passar por definir uma chave de encriptação do viewstate comum para os dois ambientes. Para mais detalhes consultar, http://msdn.microsoft.com/en-us/library/ms998288.aspx [Configure MachineKey in ASP.NET 2.0]
- Problema do header Connection: keep-alive
- Não é garantida a execução dos pedidos pela ordem correcta. Não está implementado.

6 Log (Nuno Sousa)

O módulo *Log* pretende, como o próprio nome indica, fazer o *Log* de pedidos e respostas realizados por qualquer aplicação *Web*.
À realização deste módulo, colocavam-se inicialmente três questões: a primeira, como fazer a captura dos pedidos e respostas, a segunda em como aceder à informação constante nos pedidos e respostas de forma a não comprometer a eficiência do tratamento do pedido por parte da aplicação *Web* "*loggada*", e por último a forma como os dados de *log* seriam persistidos.

Para fazer a captura, foi adoptada a implementação da interface *IHttpModule*, fazendo no seu método *Init*, chamado no *pipeline*, o registo no evento a capturar, *EndRequest*. Desta forma, por qualquer resposta enviada pela aplicação *web* "*loggada*", a classe *Log* é notificada e é executado o método registado no evento *EndRequest*, *app_EndRequest*.

Resolvida a primeira questão, colocava-se o problema de não comprometer a eficiência da aplicação *web* "*loggada*" em responder aos pedidos solicitados. Convém realçar que a mesma *Thread* responsável pela execução do código nos módulos presentes no *pipeline*, é também a responsável pelo atendimento do pedido propriamente dito, daí a necessidade de eficiência, leia-se rapidez, em "despachar" a *Thread* corrente.

Por forma a conseguir a eficiência desejada, foi utilizada uma estratégia de chamada assíncrona. É realizada a construção de um objecto *Logger*, que contém, para além das propriedades a serem "*loggadas*", um *delegate* e um *AsyncCallBack* que possibilitam o recurso à chamada assincrona do seu método *StartLog*.

Por forma a garantir a persistência da informação "*loggada*", o método *StartLog* dá início à construção dum *XmlDocument*, que será posteriormente guardado, aquando da chamda ao *AsyncCallBack*, contendo toda a informação julgada relevante e passível de ser persitida.

O método chamado através do *AsyncCallBack*, *EndLog*, é o responsável por, após averiguar a correcta terminação da chamada assíncrona, persistir a informação para um ficheiro *XML*.

Resta acrescentar que para o seu funcionamento, como qualquer outro IHttpModule, é necessário fazer o seu registo no web.config da aplicação Web a "*loggar*".

```
<add name="Log" type="MineSweeperLog.Log, MineSweeperLog" />
```

7 Forum (Ricardo Neto)

7.1 Introdução

Com o objectivo de abranger grande parte dos conteúdos leccionados, decidiu-se implementar um forum.
Apesar de, o resultado final, ser um forum com poucas ou nenhumas funcionalidades, conseguiu-se atingir o objectivo pretendido, implementando-se uma plataforma que permitisse demonstrar de forma simples os seguintes conceitos:

- Folhas de Estilo
- ClientSide Scripting (Javascript e jQuery)
- Pedidos HTTP assíncronos (AJAX) e tratamento de resposta
- Roteamento (ASP.NET MVC)
- Modelbinding (ASP.NET MVC)
- Views e partial views
- Ligação com modelo de dados (LINQ to SQL)

Optou-se por não se fazer o desenvolvimento de detrerminadas funcionalidades (e.g. paginação), uma vez que, não iria trazer novidade, no que respeita à implementação total.

7.2 Descrição Funcional

O forum é iniciado a partir do momento em que o utilizador pressiona o respectivo botão, presente no menu de Lobby. Nesta altura é criado um separador que, ao ser seleccionado, irá fazer um pedido a um endpoint (Forum/Main) que devolverá a página principal do Forum. Assim que este carregamento concluir, é feito novo pedido, agora ao endpoint responsável por apresentar a lista de tópicos existentes (Forum/ListThreads).
A partir deste momento é possível ao utilizador fazer as acções de adicionar novo tópico ou de, no âmbito de um tópico, adicionar um novo post.

Os dados representados pelo forum são guardados numa base de dados SQL, sendo para tal necessário que, antes da sua utilização, seja construído o modelo físico, recorrendo ao ficheiro disponibilizado para o efeito (CreateDataStructure.sql), bem como, proceder à alteração da string de conexão nas settings projecto MinesweeperForum.

```
[global::System.Configuration.DefaultSettingValueAttribute("Data
Source=GUMMIE\\SQLEXPRESS;Initial Catalog=MSF_Forum;Integrated
Security=True")]
```

7.3 BoxModel

O modelo definido para o forum consiste na utilização de uma barra lateral que contém os controlos para a navegação no forum, bem como controlos que permitem acrescentar tópicos ou posts, e inclui também uma área de conteúdos.

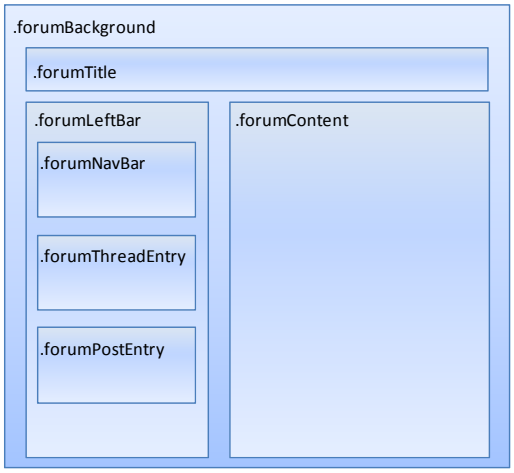


Fig. 12 - BoxModel do Forum

Toda a informação obtida do servidor será mostrada na área de conteúdo na propriedade innerhtml do elemento div correspondente. Nesta implementação apenas são possíveis dois tipos de informação: tópicos ou posts. O modelo definido para estes elementos consiste numa divisão das secções de cabeçalho, corpo e rodapé.

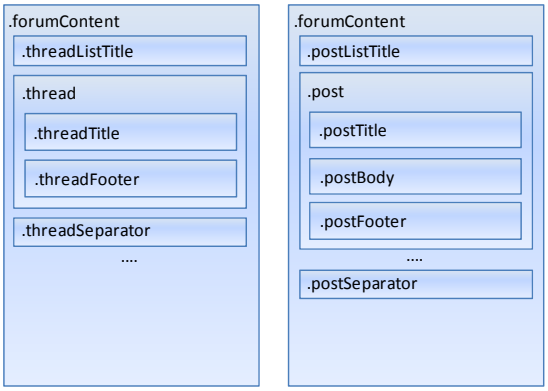


Fig. 13 - BoxModel de Threads e Posts

7.4 ClientSide

Para a implementação do lado Cliente foram criados os objectos ForumModel, ForumController e ForumView que são devidamente iniciados pela propriedade Forum::init(pName, eMail). Cada um destes objectos possui propriedades relativas ao seu papel, no âmbito da arquitectura MVC.

7.4.1 Model

ForumModel é o objecto responsável por armazenar dados a serem guardados pelo Cliente. É iniciado recebendo o nome do jogador e o seu E-mail. Um outro dado disponibilizado pelo modelo de dados é a informação acerca de qual a thread actual, a fim de se poder enviar, em conjunto com um novo post.

7.4.2 Controller

ForumController é o objecto responsável por responder às interacções do Cliente na UI e fazer chegar o resultado das mesmas ao servidor. É ele também o responsável por fazer chegar a resposta do servidor à UI.

7.4.3 View

Os aspectos visuais são manipulados pelo objecto ForumView, que possibilita funções de alteração aos componentes forumNavBar, forumThreadEntry, forumPosEntry e forumContent. Este objecto é também responsável por criar alguns elementos gráficos, e.g. botões. A esse tipo de controlos é designado um membro de ForumController para fazer o handle do evento de click.

7.5 ServerSide

A arquitectura no lado do servidor baseia-se no mesmo principio de separação de responsabilidade da arquitectura MVC.

7.5.1 Model

O modelo de dados utiliza LINQ to SQL, o que facilita não só a representação no paradigma object oriented do modelo físico, mas também todas as operações comuns de adição, remoção, actualização e consulta.
A string de conexão existe na forma de uma Foi criada uma string de conexão que, conforme indicado anteriormente, deverá ser alterada para correcta utilização e, foram também criadas 2 classes que representam tabelas e respectivas colunas.

```
[Table(Name = "Thread")]
public class Thread
{
    [Column(IsPrimaryKey = true, IsDbGenerated = true)]
    public int Id { get; private set; }
    [Column]
    public string Publisher { get; set; }
    [Column]
    public string Title { get; set; }
    [Column]
    public int Visits { get; set; }
    [Column]
    public DateTime AddDate { get; set; }
}
```

O código apresentado associa uma tabela a uma classe na aplicação. Para tal, é obrigatório classificar a classe com o atributo Table e as suas propruedades com o atributo Column. A manipulação de uma tabela envolve a criação de um DataContext, que representa as entidades mapeadas na conexão à base de dados e, através dele, obter a tabela pretendida, manipulando-a através do objecto Table.

```
DataContext dc = new DataContext(connString);
Table<Thread> threads = dc.GetTable<Thread>();
```

7.5.2 Controller

Foi criado o ForumController, sendo este o responsável por responder aos pedidos do Cliente, no âmbito do Forum, através dos seus Action Methods.
A comunicação entre Cliente e Servidor é feita, na sua maioria, através de pedidos AJAX, optando-se por pedidos do tipo GET. O ModelBinding é possível, uma vez que se manteve o mesmo nome para chaves na QueryString e parâmetros dos ActionMethods.

```
public class ForumController : Controller
{
      private string connString =
MinesweeperForum.Properties.Settings.Default.MSF_ForumConnectionString
;
```

Campo privado onde guarda informação relativa à string conexão, sendo esta utilizada em maior parte das acções.

```
      public ActionResult Main(string pName, string eMail) {…}
```

Este é o “ponto de entrada” do Forum. Através da chamada a este endpoint é devolvida toda a estrutura inicial do forum que será depois propagada com o resultado da chamada às restantes acções.

```
public ActionResult ListThreads(){…}
```

Acção responsável por devolver a lista de tópicos existentes. É atribuído ao Model de ViewData um IEnumerable<Thread>, resultado de uma pesquisa na tabela Thread da BD.


```
public ActionResult ListPosts(int? thId) {…}
```

Acção responsável por devolver a lista de posts referentes a um determinado tópico. A listagem de um tópico implica o incremento do seu número de visitas, sendo o mesmo realizado nesta acção. É atribuído ao Model de ViewData um Ienumerable<Post>, resultado de uma pesquisa na tabela Post da BD, onde o tópico pai é dado pelo parâmetro thId.

```
public ActionResult AddThread(string thTitle, string eMail) {…}
```

Acção responsável por adicionar um tópico. Recebe o título do tópico e o Email de quem o pretende inserir. Caso já exista um tópico com o mesmo título, não ocorre inserção e é enviada uma resposta de texto com o título a inserir. Caso contrário, é feita a inserção e é enviada uma resposta de redirect (302 Found) para a acção ListThreads.

```
public ActionResult AddPost(int? thId, string body, string
eMail) {…}
```

Acção responsável por adicionar um post a um tópico. Recebe o id do tópico pai, a mensagem do post e o Email de quem o pretende inserir. Após a inserção é enviada uma resposta de redirect (302 Found) para a acção ListPosts enviando o parâmetro thId.

```
}
```

7.5.3 View

A primeiro View a ser chamada no Forum é Main, uma vez que, além de ser esta construir toda a estrutura do forum, é também esta que cria os objectos ForumModel, ForumView e ForumController no lado do Cliente, tendo que receber para tal em ViewData informação acerca do nome do jogador (ViewData[“pName”]) e do Email do jogador (ViewData[“eMail”]).
Existem 2 views parciais referentes à listagem de tópicos (ListThreads) e listagem de posts (ListPosts) que recebem em Model o IEnumerable para os tipos a que dizem respeito.

7.6 Conclusões

Após o desenvolvimento desta funcionalidade, bem como, o desenvolvimento em grupo da aplicação MinesweeperFlags, conclui-se que a camada de apresentação (CSS), apesar de não requerer conhecimento de aspectos fundamentais inerentes ao desenvolvimento Web (e.g. protocolo Http), apresenta-se como um peso adicional e que, dadas as aplicações existentes online, não deve ser descurada.
No que respeita à criação de scripts ClientSide é de destacar o poder do jQuery, permitindo não só uma fácilidade de manipulação de elementos no documento HTML, mas também a facilidade com que se conseguem aplicar efeitos (e.g. hide com fade out) ou fazer o bind de eventos, tudo isto envolvendo pouco código comparativamente à API DOM disponível no Javascript.

O desenvolvimento do lado do servidor, no que respeita à arquitectura ASP.NET MVC, é facilitada pelo facto de estarem definidas convenções que permitem uma abstração relativa à forma de como tudo se "encaixa". Nota-se também um uso extenso de reflexão em toda a arquitectura, permitindo mais uma vez facilitar todo o trabalho (e.g. bind de pares chave valor de um pedido a uma colecção parâmetro de um Controller).

No que respeita à utilização do LINQ to SQL foi uma agradável surpresa constatar a facilidade com que se consegue fazer a ligação a um modelo físico e operar sobre o mesmo utilizando expressões muito semelhantes à linguagem SQL.