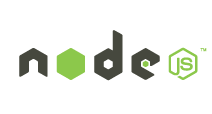


**Relatório Trabalho Final**

1. 
2. **Trabalho realizado por:**
3. Paulo Coelho nº 30076

Índice

[Introdução 4](#_Toc393143898)

[Análise do Problema 5](#_Toc393143899)

[Modelo de Dados 5](#_Toc393143900)

[Paginas HTML 6](#_Toc393143901)

[URIs de Recursos 6](#_Toc393143902)

[Plataforma de Desenvolvimento 9](#_Toc393143903)

[Microsoft ASP.NET MVC 9](#_Toc393143904)

[Node.js 9](#_Toc393143905)

[Requisitos 11](#_Toc393143906)

[ASP.NET MVC 11](#_Toc393143907)

[NODE.JS 11](#_Toc393143908)

[Implementação 13](#_Toc393143909)

[Modelo de Implementação 13](#_Toc393143910)

[Detalhe do modelo de Implementação 13](#_Toc393143911)

[Páginas HTML e CSS (Views) 14](#_Toc393143912)

[Implementação em ASP.net MVC 15](#_Toc393143913)

[Autenticação 15](#_Toc393143914)

[Visualização paginada da lista de anúncios (com informação resumida - sem detalhe) 16](#_Toc393143915)

[Visualizar detalhes do anúncio 16](#_Toc393143916)

[Adicionar comentário 16](#_Toc393143917)

[Adicionar classificação a um anúncio 16](#_Toc393143918)

[Adicionar/Editar um anúncio 16](#_Toc393143919)

[Cancelar/Remover anúncio (apenas o seu proprietário) 16](#_Toc393143920)

[Pesquisa 17](#_Toc393143921)

[Listagem dos anúncios de um utilizador 17](#_Toc393143922)

[Remoção de um anúncio por parte do utilizador que o criou 17](#_Toc393143923)

[Implementação em Node.Js 17](#_Toc393143924)

[Criação da instancia para o Servidor 17](#_Toc393143925)

[Autenticação 18](#_Toc393143926)

[Error Handling 18](#_Toc393143927)

[Controllers 19](#_Toc393143928)

[Visualização paginada da lista de anúncios 19](#_Toc393143929)

[Visualizar detalhes do anúncio 19](#_Toc393143930)

[Adicionar comentário 19](#_Toc393143931)

[Adicionar classificação a um anúncio 19](#_Toc393143932)

[Adicionar/Editar um anúncio 20](#_Toc393143933)

[Cancelar/Remover anúncio (apenas o seu proprietário) 20](#_Toc393143934)

[Pesquisa 20](#_Toc393143935)

[Existência de utilizadores na aplicação, com as respectivas operações que lhe estão associadas, nomeadamente Registo de utilizadores e Login 20](#_Toc393143936)

[Listagem dos anúncios de um utilizador 21](#_Toc393143937)

[Remoção de um anúncio por parte do utilizador que o criou 21](#_Toc393143938)

[Preenchimento de todas as informações que dependam de selecções anteriores 21](#_Toc393143939)

[Envio de email 21](#_Toc393143940)

[Terminar um anuncio 21](#_Toc393143941)

[Conclusão 22](#_Toc393143942)

[Software 23](#_Toc393143943)

[Bibliografia e Referências 24](#_Toc393143944)

Índice de figuras

[Figura 1 - Modelo de dados 5](#_Toc392851882)

[Figura 2 - Exemplo REST 7](#_Toc392851883)

[Figura 3 - Lista dos URIs 8](#_Toc392851884)

[Figura 4 – Funcionamento da plataforma Nodejs. 10](#_Toc392851885)

[Figura 5 - Modelo de Implementação 13](#_Toc392851886)

[Figure 6 - Modelo Detalhado 14](#_Toc392851887)

# Introdução

No âmbito da disciplina *Programação de Internet*, foi desenvolvida uma aplicação web para a gestão de anúncios de automóveis.

Com esta aplicação será possível pesquisar por diversos anúncios, de acordo com as propriedades que definem um anúncio. Quando a pesquisa é efectuada, é apresentada uma lista de Anúncios (de acordo com o resultado de pesquisa), na qual podemos visualizar o detalhe.

Se o utilizador estiver autenticado, caso não seja o proprietário do anuncio, poderá efectuar comentários ao mesmo sendo posteriormente enviado um e-mail ao proprietário com a informação do anuncio.

As tecnologias utilizadas foram HTML, CSS, JavaScript, Jquery, ASP.Net MVC e NodeJS. O protocolo de comunicação utilizado foi HTTP (Hyper Text Transfer Protocol).

# Análise do Problema

Após a leitura e compreensão do enunciado começou por se fazer o modelo de dados e de seguida o esquema das páginas que a serem criadas para a resolução do problema bem como a lista dos métodos utilizados para a obtenção de páginas, obtenção e criação de dados no modelo de dados.

## Modelo de Dados

De acordo com o enunciado do trabalho, e após análise do modo do funcionamento da aplicação, foi criado o seguinte modelo de dados para permitir a persistência dos dados utilizados na aplicação.

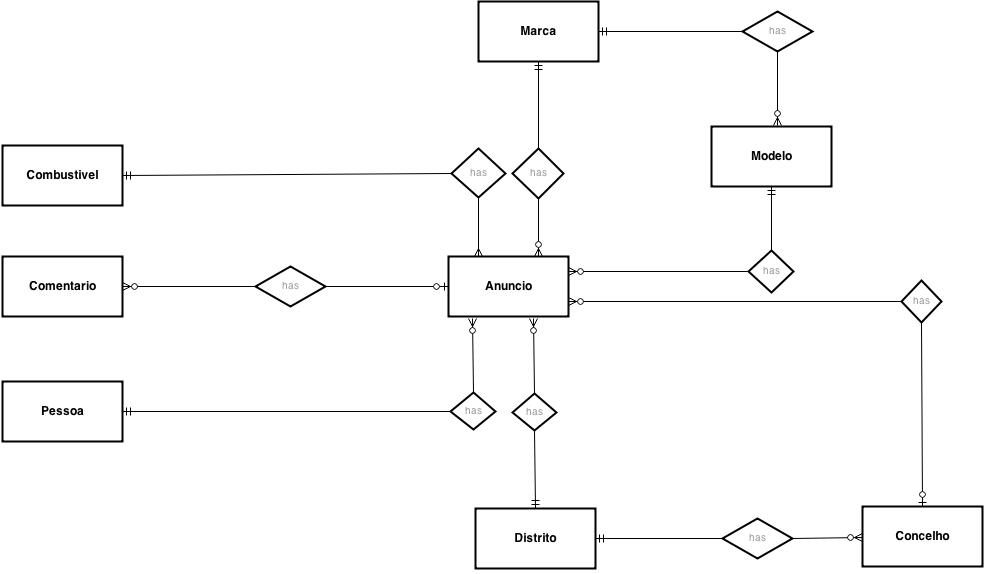


Figura 1 - Modelo de dados

De forma mais detalhada, cada uma das entidades terá os seguintes campos:

**Anuncio** (id, titulo, marca, modelo, versao, ano, km, combustivel, distrito, concelho, anunciante, preco, negociavel, validade, terminado)

**Combustivel** (id, nome)

**Comentario** (id, anuncio, username, email, titulo, texto)

**Concelho** (id,nome, idDistrito)

**Distrito** (id, nome)

**Marca** (id, nome)

**Modelo** (id,nome, idMarca)

**Pessoa** (id, primNome, UltNome, email, password)

**Classificacao** (anuncio, classificacao, username)

Devido ao facto de não ser um requisito a implementação do modelo de dados numa base de dados, os dados base são previamente inseridos aquando da criação de objectos de cada tipo e inseridos no repositório.

## Paginas HTML

Após a elaboração do modelo de dados, foi criado a lista das páginas HTML a serem utilizadas na solução bem como o fluxo da interacção entre as mesmas (e.g. (paginaCurrente, {outraPagina1,outraPagina2, …}):

(**Inicio**,{Sobre,PesquisaAvancada,Registar,Entrar,Sair,Perfil})

(**Sobre**,{Inicio,PesquisaAvancada,Registar,Entrar,Sair,Perfil})

(**PesquisaAvancada**,{Inicio,Sobre,Registar,Entrar,Sair,Perfil,Detalhe})

(**Registar**,{Inicio,Sobre,PesquisaAvancada,Entrar,Sair,Perfil})

(**Entrar**,{Inicio,Sobre,PesquisaAvancada,Registar,Sair,Perfil})

(**Sair**,{Inicio,Sobre,PesquisaAvancada,Entrar,Registar,Perfil})

(**Perfil**,{Inicio,Sobre,PesquisaAvancada,Entrar,Registar,Sair,Detalhe})

(**Detalhe**,{Inicio,Sobre,PesquisaAvancada,Registar,Entrar,Sair,Perfil,Comentar,Editar})

(**Comentar**,{Inicio,Sobre,PesquisaAvancada,Registar,Entrar,Sair,Perfil,Detalhe})

(**Editar**,{Inicio,Sobre,PesquisaAvancada,Registar,Entrar,Sair,Perfil})

## URIs de Recursos

Após a elaboração do esquema das páginas HTML, que serão apresentadas ao utilizador, foi necessário criar a lista dos uris necessários para que a interacção entre as páginas e o servidor realizassem o pretendido no enunciado.

Foi decidida a implementação da arquitectura REST (Representational State Transfer). Esta arquitectura é baseada num protocolo cliente-servidor *stateless*[[1]](#footnote-1), na qual nem o cliente nem o servidor guardam estado das comunicações entre mensagens.

As aplicações REST utilizam pedidos HTTP para as operações CRUD[[2]](#footnote-2), ou seja, para ler, enviar e remover dados. A Figura 2 exemplifica um pedido GET para obtenção de dados do servidor.

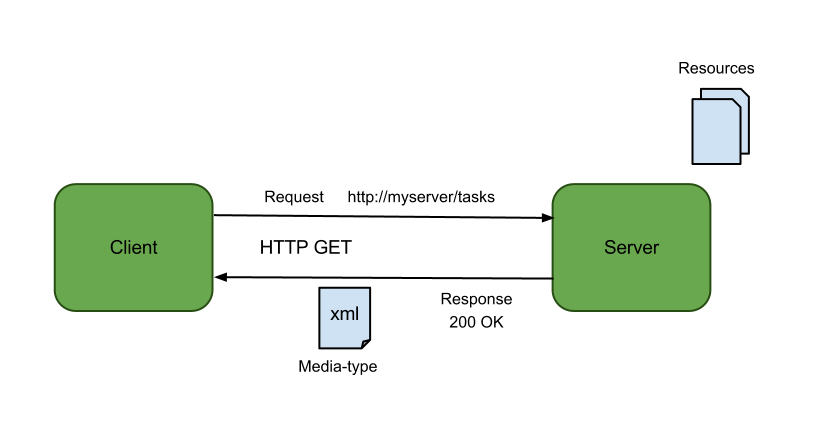


Figura 2 - Exemplo REST

Desta forma apresenta-se na Figura 3 a lista dos URI’s com os métodos utilizados e a descrição dos mesmos.

|  |  |  |
| --- | --- | --- |
| **URI** | **Método** | **Descrição** |
| / | GET | Obtem a página principal |
| /Inicio | GET | Obtem a página principal |
| /Sobre | GET | Obtem a página sobre o projecto |
| /Pessoa/Autenticar | GET | Obtem a página para autenticar um utilizador |
| /Pessoa/Autenticar | POST | Fazer Login |
| /Pessoa/Registar | GET | Obtem a página para registar um utilizador |
| /Pessoa/Registar | POST | Regista a pessoa |
| /Pessoa/Perfil/Pagina/{pagina} | GET | Permite obter a lista de anuncios de um utilizador por página |
| /Pessoa/Perfil/{id} | GET | Obtem a página de perfil de um utilizador especificado pelo id |
| /Pessoa/Sair | POST | Fazer Logout |
| /Anuncio/Adicionar | GET | Obtem a página para adicionar um Anuncio |
| /Anuncio/Adicionar | POST | Guarda um Anuncio |
| /Anuncio/Editar/{id} | GET | Obtem a página com os dados para serem editados |
| /Anuncio/Editar/{id} | POST | Permite guardar os dados editados |
| /Anuncio/Remover/{id} | POST | Remove um determinado anuncio |
| /Anuncio/Detalhe/{id} | GET | Obtem um anuncio especificado pelo id |
| /Anuncio/Comentar/{id} | GET | Permite obter a página de comentários |
| /Anuncio/Comentar/{id} | POST | Permite efectuar um comentário a um anuncio |
| /Anuncio/Comentar/Remover/{id} | POST | Permite remover um comentário de um anuncio específico |
| /Anuncio/Pesquisa | GET | Obtem a página de Pesquisa |
| /Anuncio/Pesquisa?type=&value= | GET | Obtem a lista dos Anuncios após ser feito Pesquisa |
| /Anuncio/Terminar/{id} | POST | Permite colocar um anuncio como terminado |
| /Anuncio/Detalhe/{id}/Classificar | POST | Permite escrever uma classificação |
| /Anuncio/Pesquisa/Pagina/:pagina | GET | Permite obter a lista de anuncios, após pesquisa, por página |
| /Anuncio/Detalhe/Pagina/:pagina | GET | Permite obter a lista de comentários de um anuncio por página |
| /Anuncio/Adicionar/Modelos/{idmarca} | GET | Permite obter os modelos associados a uma marca |
| /Anuncio/Adicionar/Concelhos/{iddistrito} | GET | Permite obter os concelhos associados a um distrito |

Figura 3 - Lista dos URIs

# Plataforma de Desenvolvimento

## Microsoft ASP.NET MVC

A Microsoft ASP.NET MVC é uma plataforma para a criação de aplicações web que usa o padrão de desenho Model-View-Controller. O padrão separa a aplicação em 3 componentes principais: MODEL que implementam a lógica para os dados de domínio da aplicação;

VIEW que exibem a interface para o utilizador da aplicação

CONTROLLER que lida com a interacção do utilizador, trabalham com o MODEL e seleccionam a View de retorno ao utilizador.

## Node.js

A plataforma escolhida para implementar o código do lado do Servidor foi o Node.js. O NodeJs é uma plataforma de Software para aplicações de rede escaláveis[[3]](#footnote-3) e cujo código é implementado em javascript. Para ser executado, é necessário ser interpretado fazendo uso do *Google’s V8 VM* (o mesmo ambiente de execução para javascript utilizado pelo *Google Chrome*).

Nos sistemas tradicionais, cada ligação (request) gera uma nova thread ocupando memória RAM do Sistema. O Node.js opera numa single-thread com suporte para centenas de ligações concorrentes usando chamadas no-blocking.

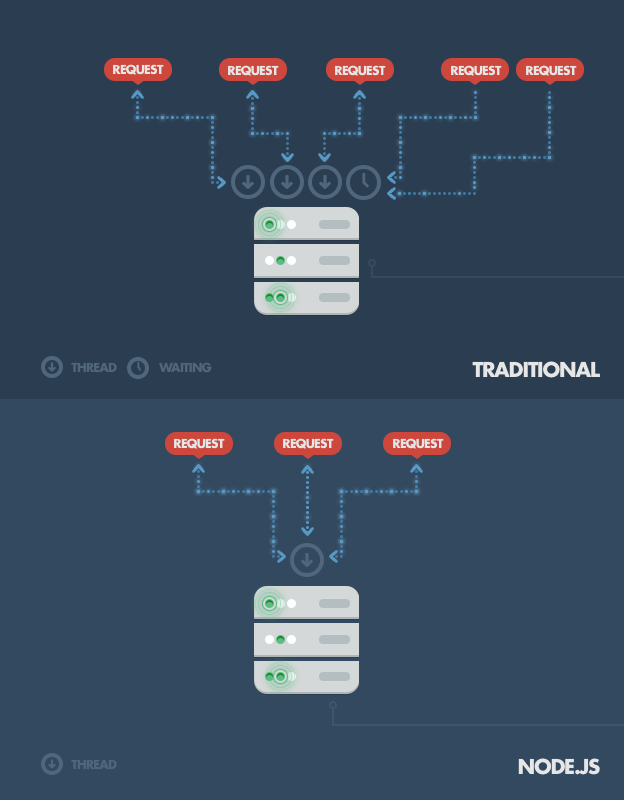


Figura 4 – Funcionamento da plataforma Nodejs.

# Requisitos

## ASP.NET MVC

Para que o projecto em ASP.NET funcione correctamente é necessário ter instalado:

* Visual Studio 2013

## NODE.JS

Para que o projecto em Node.Js funcione correctamente é necessário ter instalado:

* Nodejs
* body-parser
* connect-multiparty
* cookie-parser
* ejs
* express
* express-session
* gravatar
* linq
* mkdirp
* nodemailer

E como editor de texto pode-se utilizar:

* Notepad++
* JetBrains WebStorm
* Sublime Text Editor

Os *browsers* aconselhados para visualizar correctamente o resultado da aplicação são:

* Google Chrome
* Mozilla Firefox

# Implementação

## Modelo de Implementação

O modelo geral de implementação segue o padrão de 3 camadas no qual existem as páginas HTML que fazem pedidos para o servidor e recebem as respostas do mesmo. O Servidor interage com uma plataforma que mantém os dados persistentes (e.g. uma base de dados).

A Figura 5 demonstra o modelo descrito anteriormente.- Modelo de Implementação

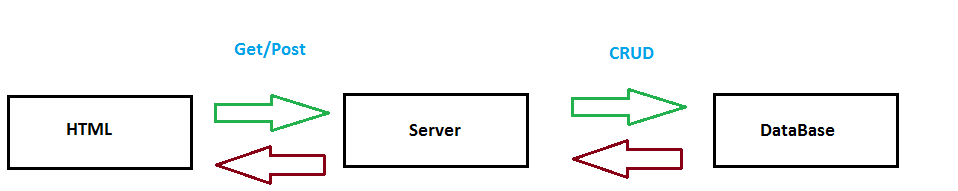


Figura 5 - Modelo de Implementação

## Detalhe do modelo de Implementação

A implementação seguiu o padrão MVC e o padrão das 3 camadas:

O Padrão MVC permite separar a representação da informação da interacção do utilizador com a mesma.

O Padrão 3 camadas permite abstrair o negócio das outras camadas, tornando o fluxo da aplicação mais simples e distribuída. Desta forma são enviados dados para o Business Logic Layer (BL) que posteriormente os envia para o Data Access Layer (DAL) para guardar numa base de dados (Figura 6).

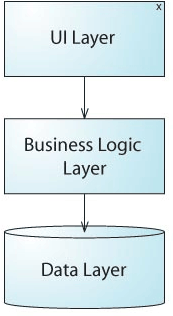


Figura 6 - Padrão 3 Camadas

Desta forma foram usados os 2 padrões para implementar a solução sendo que o Model é o mesmo para os 2 padrões:

* É apresentado ao utilizador as Views necessárias para a interacção com o servidor.
* O Controller recebe os pedidos do cliente e envia os dados para a BL que posteriormente vai enviar para a DAL de forma a obter ou guardar os dados.
* O DAL é implementado usando um repositório que contém as operações CRUD[[4]](#footnote-4), sendo que o modo como o DAL é implementado pode ser futuramente alterado sem ser necessário alterar as outras camadas.

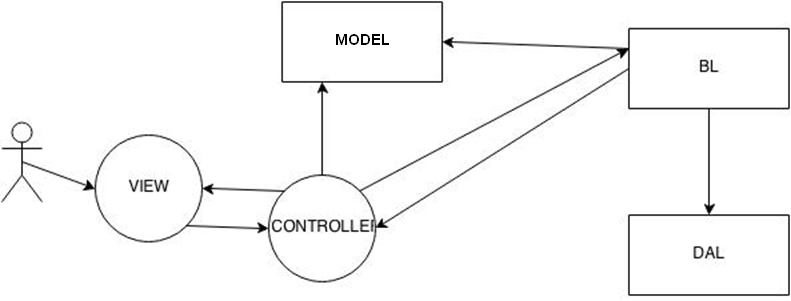
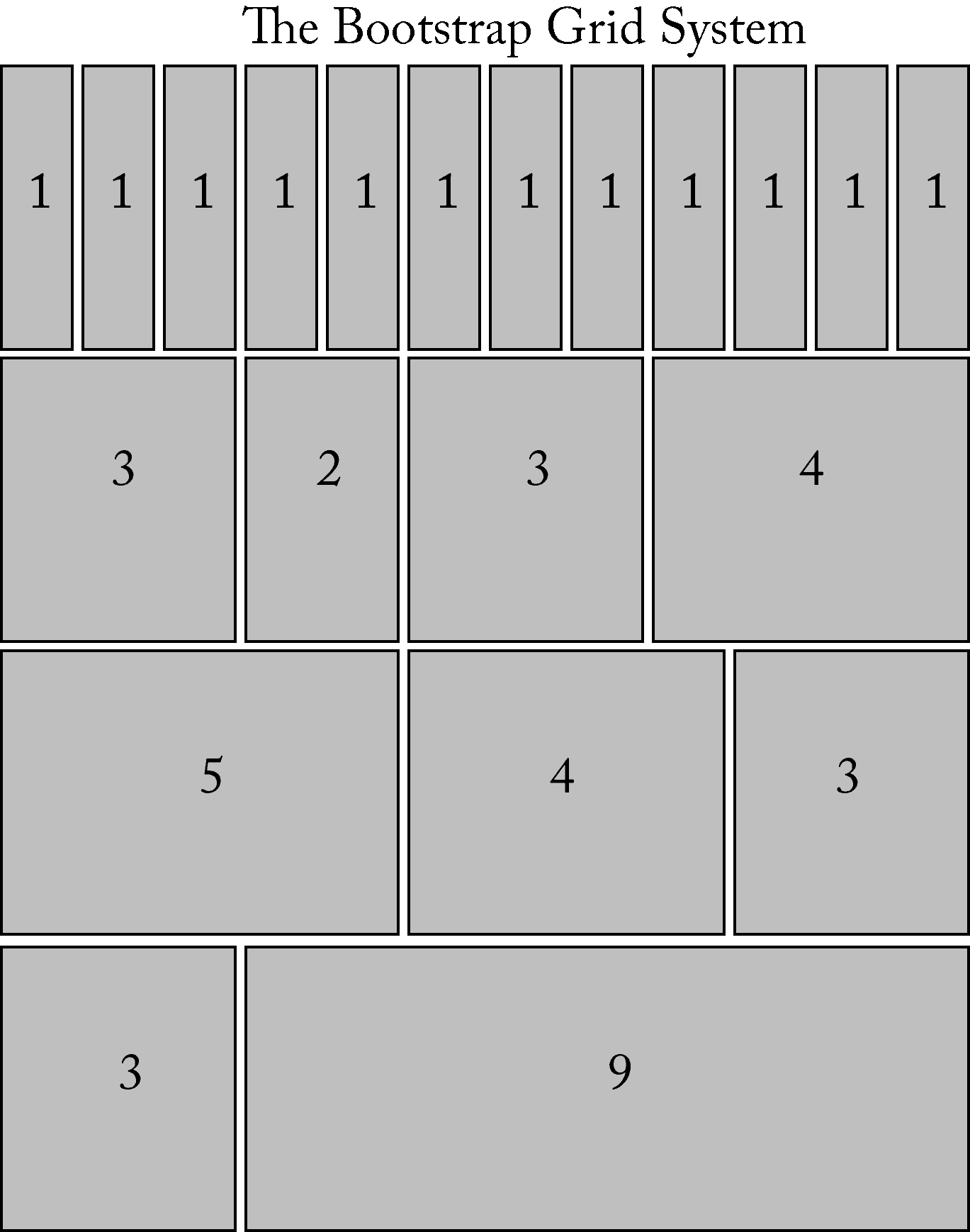


Figure 7 - Modelo Detalhado

## Páginas HTML e CSS (Views)

As páginas utilizadas no trabalho foram desenhadas usando os modelos do Bootstrap.

Desta forma tornou-se mais fácil a criação de páginas cuja apresentação fosse atractiva embora se perca um pouco a noção do modo de funcionamento de algumas funcionalidades do CSS. Uma das maiores dificuldades apresentadas nesta fase foi a compreensão do modo do funcionamento do sistema de grelha do bootstrap.





# Implementação em ASP.net MVC

### Autenticação

Para realizar a autenticação e filtrar os métodos que só podem ser acedidos quando um utilizador estiver autenticado foi criado um Filtro de Autorização para o efeito.

O filtro reencaminha o utilizador para a página de Login.

Quando é feito login é criado um cookie:

FormsAuthentication.SetAuthCookie(model.Email, model.RememberMe);

Isto adiciona um novo cookie utilizado para a autenticação do utilizador. Este novo *cookie* é independente do cookie utilizado para armazenar informações da sessão. O primeiro parâmetro é referente ao que identifica o utilizador. O segundo parâmetro é um booleano relativo ao tipo do *cookie*, se é permanente ou não. Caso seja true, ele sempre irá considerar que o utilizador está autenticado após a primeira autenticação.

### Visualização paginada da lista de anúncios

Quando é realizado uma pesquisa é retornado uma lista só com os anúncios da primeira página. Quando é solicitado a troca de página, é enviado o número da página solicitada através do uri **/Anuncio/Pesquisa?type=ano&value=2003&value=2015&page=1** e é retornado um html com os novos dados.

### Visualizar detalhes do anúncio

Caso o utilizador pesquise um anuncio, ou caso indique o url no browser, é retornado uma página com os dados do anuncio carregados bem como a lista paginada de comentários efectuados nesse anuncio.

Caso o utilizador esteja logado e seja o criador do anuncio, é-lhe mostrado o botão de Editar e Terminar (que serão descritos posteriormente).

Caso o utilizador esteja logado mas não seja o criador do anuncio, é-lhe mostrado o botão de Comentar de forma a poder adicionar um comentário ao anuncio.

### Adicionar comentário

Quando se pretende adicionar um comentário, é aberto uma página com as textboxes para escrever um comentário. Posteriormente é enviado o comentário para o servidor, voltando de seguida para a página do Anuncio.

Só pode adicionar comentário quem estiver autenticado e não for dono do anúncio.

### Adicionar classificação a um anúncio

A classificação é feita através de um elemento html Select que contém os valores 0 a 5.

Após ser classificado, uma *label* mostra o valor médio da classificação do anúncio.

### Adicionar/Editar um anúncio

A página de Adicionar retorna uma página com as textbox vazias para serem preenchidas e criadas um novo anúncio.

A página de Editar retorna uma página com os dados preenchidos previamente.

Foi usado uma *Partial View* de forma a facilitar a não repetição de código.

De forma a facilitar os testes não é obrigatório ter todos os dados preenchidos para adicionar/editar um anúncio.

### Cancelar/Remover anúncio (apenas o seu proprietário)

Caso um utilizador tente remover um anuncio ocorrem 3 situações:

1. Se o utilizador não estiver autenticado, retorna 401 e é feito redirect para a página de autenticação.
2. Se o utilizador estiver autenticado e não fôr o proprietário do anúncio é retornado 403 e a página a indicar que não tem permissões para realizar a acção pretendida.
3. Se o utilizador estiver autenticado e for o proprietário do anúncio, retorna 200 e é redireccionado para a página de perfil.

### Pesquisa

Quando o utilizador selecciona um dos dados que pretende pesquisar, a queryString da pesquisa fica com o valor **?type=tipoSeleccionado&value=valorIntroduzido&type=&value=…**.

Do lado do servidor o modo de pesquisa vai verificar qual o tipo selecionado utilizando linq.

(from res in allResults

where (!string.IsNullOrWhiteSpace(titulo) && res.Titulo != null && res.Titulo.Contains(titulo)) ||

(!string.IsNullOrWhiteSpace(marca) && res.NomeMarca != null && res.NomeMarca.Contains(marca)) ||

(!string.IsNullOrWhiteSpace(modelo) && res.NomeModelo != null && res.NomeModelo.Contains(modelo)) ||

(!string.IsNullOrWhiteSpace(combustivel) && res.NomeCombustivel != null && res.NomeCombustivel.Contains(combustivel)) ||

(!string.IsNullOrWhiteSpace(distrito) && res.NomeDistrito != null && res.NomeDistrito.Contains(distrito)) ||

(!string.IsNullOrWhiteSpace(concelho) && res.NomeConcelho != null && res.NomeConcelho.Contains(concelho)) ||

(ano != 0 && res.Ano == ano) ||

(versao != 0 && res.Versao == versao) || (preco != 0 && res.Preco == preco)

select res).ToList();

### Listagem dos anúncios de um utilizador

Ao entrar na página de perfil de um utilizador encontram-se, de forma paginada, todos os anúncios do utilizador.

### Remoção de um anúncio por parte do utilizador que o criou

Caso na página de perfil o utilizador que acede seja o proprietário do perfil, pode remover os anúncios dele, sendo imediatamente removido da página de perfil.

# Implementação em Node.Js

A implementação em Node.js (que também seguiu o padrão MVC) apresentou algumas vantagens em relação ao ASP.NET MVC, nomeadamente o Roteamento, no qual as rotas são previamente definidas por nós o que torna mais fácil de atribuir as funções a cada rota.

### Criação da instancia para o Servidor

Inicialmente são definidos os módulos http e express que instanciam o servidor.

http.createServer(app).listen(app.get('port'), function(){

console.log('Server Running on http://127.0.0.1:' + app.get('port'));

});

Posteriormente são chamados os Controllers e as funções de cada Controller são associados às rotas.

function init(app,checkAuth,AnuncioController) {

app.get('/Anuncio/Adicionar',checkAuth,AnuncioController.adicionarGet);

app.post('/Anuncio/Adicionar',multipartMiddleware,AnuncioController.adicionarPost);

…

}

De forma a se conseguir enviar ficheiros para o servidor é necessário recorrer ao middleware connect-multiparty.

### Autenticação

De forma a garantir que existem páginas que só podem ser acedidas quando o utilizador está autenticado, foi criada a função checkAuth.

A função checkAuth verifica se existe em sessão o utilizador que pretende aceder e caso não exista, guarda em sessão o uri que pretendia aceder, retorna o código 401 e a página de login. Após o utilizador se autenticar, é retornada a página que o utlizador pretendia aceder.

Poderia ter sido implementado usando Simple\_authentication, enviando um header WWW-Authenticate na resposta para o utilizador se autenticar mas optou-se por esta abordagem.

function checkAuth(request, response, next) {

if (!request.session || !request.session.user)

{

request.session.pre\_url = request.originalUrl;

var err = new Error();

err.status = 401;

next(err);

} else {

next();

}

}

### Error Handling

Foi criado um Controller que contém os métodos para o tratamento de erros que possam ocorrer.

Os erros tratados são os que contém o código **400** (Bad Request), **401** (Unauthorized), **403** (Forbidden) e **404** (Not Found). É retornado uma página a indicar qual o erro ocorrido com 1 botão para retornar à página de Inicio.



### Controllers

Os Controllers contêm métodos que recebem como parâmetro o request e o response sendo que os pedidos são tratados na BL que por sua vez acedem ao Repositório que, conforme descrito anteriormente, trata os dados e retorna os mesmos para que sejam enviados no Response.

### Visualização paginada da lista de anúncios

Quando é realizado uma pesquisa é retornado uma lista só com os anuncios da primeira página. A paginação é feita por Ajax. Quando é solicitado a troca de página, é enviado o número da página solicitada e é retornado um html com os novos dados.

### Visualizar detalhes do anúncio

Caso o utilizador pesquise um anuncio, ou caso indique o url no browser, é retornado uma página com os dados do anuncio carregados bem como a lista paginada de comentários efectuados nesse anuncio.

Caso o utilizador esteja logado e seja o criador do anuncio, é-lhe mostrado o botão de Editar e Terminar (que serão descritos posteriormente).

Caso o utilizador esteja logado mas não seja o criador do anuncio, é-lhe mostrado o botão de Comentar de forma a poder adicionar um comentário ao anuncio.

### Adicionar comentário

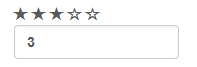
Quando se pretende adicionar um comentário, uma popup é aberta com as textboxes para escrever um comentário. Posteriormente é enviado o comentário para o servidor, voltando de seguida para a página do Anuncio.

Só pode adicionar comentário quem estiver autenticado e não for dono do anuncio.

### Adicionar classificação a um anúncio

A classificação é feita por Ajax usando imagem de estrelas que contém o valor de 1 a 5.

Após ser classificado, uma *label* com estrelas mostra o valor médio da classificação do anuncio.



Quando é feito o carregamento da página que contém as estrelas são adicionados 2 eventos:

* ***mouseover*** que altera o *css* da estrela quando se coloca o mouse por cima dele.

stars[i].addEventListener("mouseover",function(){

var found = false;

for(var j=0;j<stars.length;++j)

{

if(!found)

document.getElementsByName("rateStar")[j].className="glyphicon glyphicon-star";

else

document.getElementsByName("rateStar")[j].className="glyphicon glyphicon-star-empty";

if(stars[j] === this)

found = true;

}

});

* ***Click*** que chama envia por Ajax o índice da estrela que foi clicada, sendo desta forma enviado a classificação para o servidor.

stars[i].addEventListener("click",function(){

AjaxRating(this.getAttribute("index"));

});

### Adicionar/Editar um anúncio

A página de Adicionar retorna uma página com as textbox vazias para serem preenchidas e criadas um novo anúncio.

A página de Editar retorna uma página com os dados preenchidos previamente.

De forma a facilitar os testes não é obrigatório ter todos os dados preenchidos para adicionar/editar um anúncio.

### Cancelar/Remover anúncio (apenas o seu proprietário)

Caso um utilizador tente remover um anuncio ocorrem 3 situações:

1. Se o utilizador não estiver autenticado, retorna 401 e é feito redirect para a página de autenticação.
2. Se o utilizador estiver autenticado e não fôr o proprietário do anúncio é retornado 403 e a página a indicar que não tem permissões para realizar a acção pretendida.
3. Se o utilizador estiver autenticado e for o proprietário do anúncio, retorna 200 e é redireccionado para a página de perfil.

### Pesquisa

Quando o utilizador selecciona um dos dados que pretende pesquisar, a queryString da pesquisa fica com o valor **?type=tipoSeleccionado&value=valorIntroduzido**, sendo que caso o tipoSeleccionado tenha 2 valores a introduzir, a queryString fica com o valor **?type=tipoSeleccionado&value=valorIntroduzido1&value=valorIntroduzido2**.

Do lado do servidor o modo de pesquisa vai verificar se os parametros são do tipo Array (se for. Caso seja o caso, significa que está a ser realizada uma pesquisa baseada em 2 parametros sendo por isso criada uma expressão que verifique as condições que estejam de acordo com a condição de pesquisa.

Se o valor do type for igual à propriedade do objecto que contêm a função de pesquisa associada e filtra os resultados.

### Existência de utilizadores na aplicação, com as respectivas operações que lhe estão associadas, nomeadamente Registo de utilizadores e Login

Quando é pretendido registar um utilizador, é enviado no request os dados do novo utilizador. Caso o utilizador já exista é enviada uma mensagem a informar que o mesmo já existe. Caso não exista é enviado o código 201 e é enviado uma mensagem de sucesso. Caso contrário é enviado o código 409 com uma mensagem de erro.

Caso o utilizador pretenda fazer Login é verificado se o utilizador existe, e nesse caso é guardado em sessão o utilizador sendo posteriormente redireccionado para a página que pretendia aceder.

### Listagem dos anúncios de um utilizador

Ao entrar na página de perfil de um utilizador encontram-se, de forma paginada e utilizando ajax, todos os anuncios do utilizador.

### Remoção de um anúncio por parte do utilizador que o criou

Caso na página de perfil o utilizador que acede seja o proprietário do perfil, pode remover os anuncios dele, sendo imediatamente removido da página de perfil.

### Preenchimento de todas as informações que dependam de selecções anteriores

Quando é seleccionado uma Marca, é feito um pedido Ajax ao Servidor para obter todos os modelos associados à Marca. Do lado do servidor é retornardo um objecto JSON com os dados, sendo posteriormente adicionados ao Select.

Quando é seleccionado um Distrito, é feito um pedido Ajax ao Servidor para obter todos os concelhos associados ao Distrito. Do lado do servidor é retornardo um objecto JSON com os dados, sendo posteriormente adicionados ao Select correspondente.

### Envio de email

Quando é efectuado um comentário, é enviado um e-mail ao proprietário do anuncio com a indicação de que um novo comentário foi colocado nesse anúncio. É também indicado o uri para o anúncio. Foi utilizado o módulo nodemailer para o efeito.

### Terminar um anuncio

Quando o dono do anuncio decidir terminar o anuncio, o registo fica actualizado com a data actual e um dos campos do registo, denominado terminado fica com o valor true. Desta forma na pesquisa irão aparecer todos os anuncios cuja data de validade esteja de acordo com a data calculada através do valor N definido no ficheiro de configuração da aplicação.

Foi adicionada ao prototype do objecto Date a função subbDays que subtrai dias à data que o chama.

Date.prototype.subbDays = function(days)

{

this.setDate(this.getDate()-days);

}

# Conclusão

Com este trabalho foi permitido aprofundar o modo de funcionamento do protocolo HTTP (estudado na disciplina) bem como o modo geral de funcionamento das aplicações WEB usando o protocolo mencionado anteriormente.

Ao utilizar o *nodejs* como plataforma para o desenvolvimento do código do servidor, foram aprofundados conhecimentos de *javascript* bem como a exploração de alguns detalhes do modo de funcionamento desta plataforma.

Foi encontrada uma dificuldade inicial na compreensão do *nodejs*, nomeadamente a linguagem *javascript* e as suas características.

Ao utilizar o ASP.NET MVC como plataforma de desenvolvimento do código do servidor, foram adquiridos conhecimentos do funcionamento desta plataforma bem como foram aplicados alguns conhecimentos obtidos na disciplina Ambientes Virtuais de Execução.

# Software

* Microsoft Visual Studio 2013
* Microsoft .Net Framework 4.5
* Node.js
* Google Chrome
* Mozilla Firefox
* JetBrains WebStorm 7
* NotePad++

# Bibliografia e Referências

<http://nodejs.org/>

<http://expressjs.com/>

http://www.w3schools.com/

<http://getbootstrap.com/>

<http://bootsnipp.com/>

<https://www.atlassian.com/git/tutorial>

<http://rubular.com/>

<http://devdocs.io/>

<http://www.codecademy.com/tracks/javascript>

<http://www.yourinspirationweb.com/en/do-you-want-to-use-json-but-dont-know-where-to-start/>

<http://www.hacksparrow.com/handle-file-uploads-in-express-node-js.html>

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

http://rest.elkstein.org/2008/02/what-is-rest.html

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

https://www.draw.io/

RESTful Service Best Practices, Todd Fredirch

The Node beginner book, Manuel Kiessling

Node up and Running, Tom Hughes-Croucher & Mike Wilson

Desenvolvimento Web com HTML, CSS e Javascript, K19

WebStorm\_ReferenceCard\_70

Eloquent JavaScript A Modern Introduction to Programming, Marijn Haverbeke

HTTP Pocket Reference, [Clinton Wong](http://shop.oreilly.com/product/9781565928626.do#tab_04_0)

[K32 - Desenvolvimento Web com ASP.NET MVC](http://www.k19.com.br/cursos/desenvolvimento-web-com-aspnet-mvc)

1. Stateless – sem estado. [↑](#footnote-ref-1)
2. CRUD – Create, Read, Update, Delete. [↑](#footnote-ref-2)
3. Escalável - aumento automático da capacidade de armazenamento ou processamento. [↑](#footnote-ref-3)
4. CRUD – Create Read Update Delete [↑](#footnote-ref-4)