



Prova de Aptidão Profissional

Redes de Comunicação

Relatório ShopTek

Turma 233 TGPSI	Nº100344	Fábio dos Santos Carvalho
------------------------	-----------------	---------------------------

Relatório ShopTek

Prova de Aptidão Profissional

Ano Lectivo 2010/2011

Turma 233 TGPSI	Nº100344 Fábio dos Santos Carvalho
------------------------	---



Técnico de Gestão e Programação de Sistemas de
Informação

Julho 2011

Coordenador de curso: Manuel Tróia

Agradecimentos

- Colegas – Por me darem apreciações gerais durante o desenvolvimento do projecto;
- Professores Manuel Tróia e Paula Graça – Pela ajuda na resolução de problemas pontuais.

Resumo

No 3º ano do curso de TGP SI, todos os alunos devem realizar um projecto representativo das competências adquiridas.

Foi pedido aos alunos a realização de um projecto no qual implementassem uma aplicação *web* utilizando a *framework* ASP.Net. Todo o projecto deveria apresentar a complexidade necessária para abranger grande parte dos conteúdos leccionados. Entre eles, o desenvolvimento de uma base de dados em *SQL Server* ou a implementação de uma *template CSS*.

Decidi, para a Prova de Aptidão Profissional do ano lectivo 2010/2011, desenvolver uma aplicação *web* de uma loja de produtos relacionados com equipamento audiovisual, videojogos, informática e fotografia.

A razão para esta escolha prende-se ao facto de ser um tipo de aplicação solicitado no mercado que, mais tarde, me irá acolher. Para além disso, já tinha ideias e imaginava as páginas do *website* e todo o seu funcionamento, o que me deu ânimo para avançar com a ideia, pois as ideias pareceram-me muito boas.

Para desenvolver este projecto, atravessei as principais fases de desenvolvimento, por ordem: Proposta, Análise, Implementação e Documentação.

Primeiro, pensei no tema do projecto e nas funcionalidades principais, seguido do planeamento, a fase de implementação e por fim os testes.

As situações mais relevantes de cada uma das fases e sub-fases são, aqui, relatadas.

Índice

1. <i>Introdução</i>	1
a. Objectivos do Projecto	2
b. Proposta do Projecto.....	4
c. Tecnologias e Recursos Utilizados.....	6
d. Contributos do Projecto.....	7
e. Organização do Relatório	8
2. <i>Realização do Projecto</i>	9
f. Fase 1: Planeamento, Proposta e Pontos de Situação.....	9
I. Escolha do tema	9
II. Objectivos.....	10
III. Planeamento das fases de desenvolvimento	10
IV. Proposta	11
V. Pontos de Situação.....	11
g. Fase 2: Análise	12
I. Planeamento do modelo de dados	12
II. Planeamento de algumas funcionalidades	17
h. Fase 3: Implementação.....	18
I. Implementação da interface	18
II. Implementação do Javascript.....	21
III. Implementação da autenticação	23
IV. Implementação da estrutura de classes	27
V. Implementação da DAL e BLL.....	32
VI. Implementação de <i>Web User Controls</i>	39
VII. Implementação do back office e notificações.....	41
3. <i>Versões</i>	43
i. Versão 1.0	43
j. Versão 1.1	43

4. <i>Conclusões</i>	44
k. Objectivos realizados	44
l. Apreciação final	47

Índice de Figuras

<i>Figura 1 – Modelo E.A. da base de dados.....</i>	<i>14</i>
<i>Figura 2 – Modelo relacional da base de dados.</i>	<i>16</i>
<i>Figura 3 – Planeamento da classe Carrinho.</i>	<i>17</i>
<i>Figura 5 – Imagem de efeito do menu.....</i>	<i>21</i>
<i>Figura 4 – Efeitos visuais.</i>	<i>21</i>
<i>Figura 6 – Classe Autenticação.....</i>	<i>23</i>
<i>Figura 7 – Classe Utilizador.</i>	<i>27</i>
<i>Figura 8 – Classe Carrinho.</i>	<i>28</i>
<i>Figura 9 – Classe Produto.</i>	<i>31</i>
<i>Figura 10 – Camadas de acesso a dados.....</i>	<i>32</i>
<i>Figura 11 – Data Access Layer.</i>	<i>33</i>
<i>Figura 12 – Business Logic Layer.....</i>	<i>36</i>
<i>Figura 13 – Adicionar um web user control pelo web.config.</i>	<i>39</i>
<i>Figura 14 – Web User Control 'ProdutoDestaque'.</i>	<i>40</i>
<i>Figura 15 – Template do back office.....</i>	<i>41</i>
<i>Figura 16 – Notificação de inserção de marca.</i>	<i>41</i>
<i>Figura 17 – Classe Notificacao.....</i>	<i>42</i>

Índice de Tabelas

<i>Tabela 1 – Datas relativas às fases de desenvolvimento do projecto.....</i>	<i>4</i>
<i>Tabela 2 – Cronograma</i>	<i>5</i>
<i>Tabela 3 – Tipos de utilizador</i>	<i>23</i>

Notação e Glossário

API - *Application Programming Interface;*

CSS – *Cascading Style Sheets;*

SQL – *Structured Query Language;*

DAL – *Data Access Layer;*

BLL – *Business Logic Layer;*

IDE – *Integrated Development Environment;*

Introdução

No âmbito do 3º ano do curso de TGPSI (Técnico de Gestão e Programação de Sistemas de Informação), foi pedida a realização da Prova de Aptidão Profissional que deve ser apresentada no final do ano lectivo.

Este projecto consiste no desenvolvimento de uma aplicação *web* com base nas competências adquiridas ao longo dos anos do curso. Tem como objectivo, não só representar essas competências, mas também expor os alunos ao desenvolvimento de um projecto mais complexo em *web*.

O tempo definido para a realização do projecto e respectiva documentação foi de, aproximadamente, 4 meses. Tendo começado com as propostas no final de Dezembro de 2010 até ao início de Janeiro de 2011, e a acabar em Abril. A data de entrega do projecto foi definida para o dia 11 de Abril de 2011.

O tema do projecto e as suas respectivas funcionalidades são deixadas ao critério do aluno. Decidi então desenvolver uma aplicação *web* de uma loja.

Ao longo desta Introdução, serão especificados os seguintes aspectos:

- Objectivos do projecto;
- Proposta do projecto;
- Tecnologias e recursos utilizados;
- Contributos do projecto;
- Organização deste relatório;

Objectivos do Projecto

O tema escolhido (loja em web) apresenta muitas funcionalidades que são facilmente perceptíveis e se encontram em qualquer loja na web.

O que consegui observar foi que em cada loja, haviam métodos diferentes de implementar essas funcionalidades, dependendo principalmente do conteúdo que a loja disponibiliza.

Os objectivos principais do projecto são os seguintes:

- Desenvolver um webiste que implemente uma loja;
 - Essa loja será indicada para a venda de produtos relacionados com a informática, jogos e equipamento audiovisual.
 - Será possível efectuar encomendas através do website;
 - As encomendas podem ser efectuadas após um registo por parte do utilizador, no qual este vai disponibilizar a informação necessária ao processamento da encomenda.
- Os produtos estão organizados por secções do website, sendo que cada secção promove apenas produtos que pertençam a essa mesma secção;
- O *website* será capaz de promover produtos aos utilizadores através de correio electrónico, tendo em conta as áreas mais visitadas por cada utilizador (*Newsletter*);
 - Haverá um tipo de newsletter para cada secção apenas.
 - A subscrição de um utilizador a uma newsletter é facultativa e pode subscrever-se em mais que uma.
- Capacidade de melhorar as procuras dos utilizadores através de vários critérios;
 - Procurar por data (ver artigos mais recentes/antigos), marca, classificação, etc;

- Interação entre utilizadores através de feedback nos produtos adquiridos;
 - A capacidade dos utilizadores registados comentarem e classificarem os produtos;
 - As classificações são facultativas para o feedback e o valor da classificação varia de 0 a 5 (inclusive).
- ‘*Back office*’ simples a fim de gerir o conteúdo da base de dados.
 - Inserir, alterar e remover produtos;
 - Inserir, alterar e remover métodos de pagamento, envio e estados de encomendas;
 - Inserir, alterar e remover marcas;
 - Adicionar/Remover stock aos produtos;

Os objectivos secundários do projecto são os seguintes:

- ‘*Back office*’ que apresente determinadas estatísticas, como por exemplo, as vendas de produtos de informática num determinado mês;
 - Estatísticas por tempo e secção;
- Melhoramento do sistema de procura através de um sistema de ‘tags’, permitindo encontrar produtos através de outros parâmetros de procura;
- Recomendar produtos aos utilizadores com base nas últimas visitas/compras efectuadas.

Proposta do Projecto

Na proposta do projecto foram definidos os objectivos principais (pág. 2) e depois, tendo em conta os objectivos, foram definidas as datas de início e fim de cada fase de desenvolvimento, assim como o cronograma mais detalhado.

A proposta do projecto foi entregue no dia 5 de Janeiro de 2011.

Fase	Data de Inicio	Data de Fim	Horas/Dias
Planeamento	13-12-2010	17-12-2010	120 / 5
Proposta	04-01-2011	04-01-2011	24 / 1
Análise	20-12-2010	25-01-2011	864 / 36
Implementação	01-02-2011	01-04-2011	1440 / 60
Documentação	06-01-2011	07-04-2011	264 / 11
Apresentação	01-07-2011	01-07-2011	24 / 1

Tabela 1 – Datas relativas às fases de desenvolvimento do projecto

A *Tabela 1* contém as **previsões** para as datas de início e fim de cada fase de desenvolvimento, assim como o número de horas para cada.

Embora a *Tabela 1* seja útil, é necessário um cronograma mais detalhado a fim de, visualmente, podermos interpretar o tempo definido para cada fase.

Fase		Dezembro				Janeiro				Fevereiro			
Proposta						En							
Análise	Requisitos												
	Modelo de Dados												
	Desenho Interface												
	Stored Procedures												
	Teste Base Dados												
Implementação	Interface												
	Código												
	Testes												
Documentação									En			En	

Fase		Março				Abril			
Proposta									
Análise	Requisitos								
	Modelo de Dados								
	Desenho Interface								
	Stored Procedures								
	Teste Base Dados								
Implementação	Interface								
	Código								
	Testes								
Documentação					En		En		

Legenda	
	Trabalho
En	Entrega

Tabela 2 – Cronograma

Tecnologias e Recursos Utilizados

Este projecto foi desenvolvido utilizando vários recursos e tecnologias.

➤ **Software**

A componente de programação (HTML, CSS, Javascript e C#) foi toda desenvolvida no IDE Microsoft Visual Studio 2010. No que toca à base de dados, o software Dia foi utilizado para a realização do modelo E.A. e do modelo Relacional, sendo que a base de dados foi desenvolvida no software SQL Server 2008, integrado no Microsoft Visual Studio.

Visto que o projecto se trata de uma aplicação *web*, o aspecto desta centra-se nas imagens e no CSS. As imagens do *website* foram realizadas e optimizadas utilizando o software Adobe Photoshop CS3.

- Microsoft Visual Studio 2010 Ultimate;
 - SQL Server 2008;
- Adobe Photoshop CS3;
- Dia;

➤ **Tecnologias**

Este projecto foi desenvolvido na *framework* da Microsoft ASP.NET.

A API de Javascript, jQuery foi utilizada para quase todo o *scripting* do lado cliente.

A linguagem de programação C# foi utilizada para o *scripting* do lado do servidor.

- *Framework* ASP.NET;
- API jQuery;
- Linguagem de programação C#;

Contributos do Projecto

Este projecto foi desenvolvido com o objectivo de servir de avaliação como Prova de Aptidão Profissional.

A importância da presença da Prova de Aptidão Profissional é clara, visto que foi o projecto mais importante que já desenvolvi como programador e foi o projecto no qual aprendi quase tudo o que sei acerca de desenvolvimento em *web*.

Foi-me introduzida a API jQuery, as CSS, e o Transact-SQL. Assim como aprofundei os conhecimentos de C# e o funcionamento da tecnologia ASP.NET.

Embora não tenha desenvolvido nada de novo, gostei bastante de desenvolver esta aplicação *web* e amadureci bastante como programador.

Organização do Relatório

O relatório está organizado pelas fases de desenvolvimento do projecto. Na segunda parte do relatório “Realização do Projecto”, serão descritas as situações mais relevantes de cada uma das fases.

- Fase 1: Planeamento, Proposta e Pontos de Situação
 - Nesta secção serão relatadas as situações relativas à fase de planeamento e proposta, assim como os pontos de situação que foram desenvolvidos e entregues ao longo da realização do projecto;
- Fase 2: Análise
 - Esta secção aborda a fase de análise: análise de requisitos, modelo de dados, desenho da interface, implementação das *stored procedures* e os testes à base de dados;
- Fase 3: Implementação
 - Nesta secção serão relatadas as situações relativas à fase de implementação do projecto: a interface, o código e testes.

Realização do Projecto

Nesta secção do relatório encontram-se relatadas todas as situações relevantes relativas às fases de desenvolvimento do projecto.

Fase 1: Planeamento, Proposta e Pontos de Situação

Esta fase foi constituída, por ordem de realização, nas seguintes sub-fases:

- I. Tema do projecto;
- II. Objectivos;
- III. Planeamento das fases de desenvolvimento;
- IV. Proposta;
- V. Pontos de situação;

I. Escolha do tema

O tema do projecto é uma loja na *web*. A razão para esta escolha prende-se ao facto de ser um tipo de aplicação solicitado no mercado que, mais tarde, me irá acolher.

Desde o 2º ano do curso que já planeava este tema para a PAP e por isso já tinha imaginado alguns aspectos como a interface ou algumas funcionalidades.

O projecto tem o nome de ShopTek.

II. Objectivos

Para os objectivos mais detalhados, consulte a página 2.

Razões para a escolha de algumas funcionalidades:

- Organização dos produtos por secções:
 - Isto permite uma rápida filtragem dos produtos de interesse aos utilizadores.
 - Exemplo: se um utilizador só tiver interesse nos produtos de fotografia, este acede à secção de fotografia, escondendo os outros produtos das outras secções.
 - Permite uma melhor organização da loja e promoção de produtos mais específica.
- Possibilidade dos utilizadores comentarem e classificarem produtos:
 - Pensei nesta funcionalidade pois é uma forma interessante dos utilizadores discutirem os produtos, expondo algumas funcionalidades ou características dos produtos que não são expostas apenas pelas especificações do produto.

Mais tarde, foi decidido que a funcionalidade de *Newsletter* **não** seria implementada.

III. Planeamento das fases de desenvolvimento

Para os cronogramas com as datas planeadas, consulte as páginas 4 e 5.

IV. Proposta

A proposta do projecto foi entregue, por e-mail ao coordenador de curso e coordenador de projectos do curso de TGPSSI no dia 5 de Janeiro de 2011.

A proposta abrange o Tema (I), Objectivos (II) e Planeamento das fases de desenvolvimento (III) que estão nas páginas anteriores e esta encontra anexada a este relatório.

V. Pontos de Situação

Para um maior controlo do desenvolvimento do projecto, assim como percepção da situação do projecto a certo ponto no tempo, foi-nos pedido a realização de pontos de situação. Isto é, pequenos relatórios que descrevam o estado do projecto e tudo o que foi realizado no espaço de tempo que delimita cada ponto de situação.

Foram marcados 3 pontos de situação, para os dias:

- 28 de Janeiro de 2011;
- 28 de Fevereiro de 2011;
- 28 de Março ed 2011.

Consulte a página de anexos para a visualização dos pontos de situação.

Fase 2: Análise

Na fase de análise é feito o levantamento de requisitos, dá-se a resolução teórica de alguns problemas do projecto, assim como a realização do modelo de dados desde o seu planeamento à sua concretização.

As principais situações da fase de análise foram as seguintes:

- I. Planeamento do modelo de dados;
- II. Planeamento de algumas funcionalidades;

I. Planeamento do modelo de dados

Antes de inciar o planeamento da interface, foi necessário planear a base de dados.

Dados os objectivos do projecto, elaborei um texto no qual estão especificados os requisitos do modelo de dados:

As entidades principais da loja são:

- Utilizadores;
- Produtos;
- Encomendas;
- Estrutura por secções;

➤ Utilizadores

- Cada utilizador apresenta várias características que serão úteis à loja a fim de processar encomendas (nome, morada, etc.)
- Cada utilizador tem apenas **um** carrinho e pode realizar **várias** encomendas;
- Cada utilizador registado pode classificar **vários** produtos;

- Cada utilizador possui um campo que determina o papel do mesmo no *website*. Ex: Administrador ou Cliente.
- Produtos
 - Os produtos devem conter as informações necessárias para esclarecer os utilizadores (nome, marca, especificações, etc.)
 - Os produtos podem ser classificados e comentados por qualquer e **vários** utilizadores registados.
 - Cada produto deve conter **um** campo que o situa na estrutura de secções do *website*.
 - Um produto possui apenas **uma** marca.
- Encomendas
 - As encomendas registam toda a informação necessária para identificar os produtos e a quantidade, assim como identificar o utilizador para que a informação seja persistente (nome, morada, etc.).
 - Cada encomenda possui apenas **um** método de pagamento, **um** método de envio e **um** estado.
 - Uma encomenda pode ter **vários** produtos sendo que tem **obrigatoriamente um**.
- Estrutura por Secções
 - O *website* deve estar estruturado por secções:
 - Secção;
 - Existem apenas 5 secções (Audiovisual, Informática, Música e Filmes, Fotografia e Videojogos).
 - Sub-secção;
 - Tem uma secção;
 - Tipo;
 - Tem uma sub-secção;
 - Sub-Tipo;
 - Tem um tipo;

O diagrama de Modelo Entidade-Relacionamento (MER) para o sistema de comércio eletrônico apresenta as seguintes entidades e atributos:

- Utilizador**: Email, Localidade, Nome, Apelidos, Morada, Login, Password, ID (chave primária), Tipo.
- Enviei**: DataEnvio (atributo de relacionamento).
- Newsletter**: ID (chave primária), Corpo, Data.
- Subscrição**: (Entidade de associação entre Utilizador e Newsletter).
- Encomenda**: Descrição, Morada, Nome, ID (chave primária), Data, Preço Total, Localidade, Desconto, N.º Visitas, Imagem, Stock, Descrição, DataInserção, Especificações, Comentário.
- Produto**: ID (chave primária), Preço, Quantidade, Website, Descrição, Marca, Sub-Tipo, Tipo, Inserir-se.
- Conteúdo**: Quantidade, Website, Descrição, Marca, Sub-Tipo, Tipo, Inserir-se.
- Secção**: Nome, ID (chave primária), Descrição, Inserir-se.
- MetodoEnvio**: Nome, Extras, Descrição, ID (chave primária).
- MetodoPagamento**: Nome, Extras, Descrição, ID (chave primária).
- Estado**: Nome, Descrição, ID (chave primária).
- Tag**: Nome, N.º Procuraas, ID (chave primária).
- Sub-Tipo**: Nome, ID (chave primária).
- Sub-Secção**: Nome, ID (chave primária).

As relações e as restrições de integridade são as seguintes:

- Enviei** relaciona **Utilizador** e **Newsletter** com uma restrição de integridade de 1 para muitos.
- Subscrição** relaciona **Utilizador** e **Newsletter** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **Utilizador** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **Produto** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **Conteúdo** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **MetodoEnvio** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **MetodoPagamento** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **Estado** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **Tag** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **Sub-Tipo** com uma restrição de integridade de 1 para muitos.
- Encomenda** relaciona **Sub-Secção** com uma restrição de integridade de 1 para muitos.
- Produto** relaciona **Tag** com uma restrição de integridade de 1 para muitos.
- Produto** relaciona **Sub-Tipo** com uma restrição de integridade de 1 para muitos.
- Conteúdo** relaciona **Sub-Tipo** com uma restrição de integridade de 1 para muitos.
- Conteúdo** relaciona **Sub-Secção** com uma restrição de integridade de 1 para muitos.
- Secção** relaciona **Sub-Secção** com uma restrição de integridade de 1 para muitos.

Figura 1 – Modelo E.A. da base de dados.

Após uns ajustes e após obter a versão final do modelo E.A. (Figura 1), pude avançar para a realização do modelo relacional.

Nesta altura houve a necessidade de definir os tipos dos campos das entidades. Visto que a base de dados vai ser realizada utilizando o software SQL Server 2008, investiguei um pouco acerca dos tipos de dados em SQL.

Decidi escolher os seguintes tipos de dados para os campos das tabelas (entidades):

- Auto int;
 - É utilizado nas chaves primárias das tabelas. Consiste num campo numérico inteiro com a funcionalidade de auto incrementação.
- Nvarchar(<tamanho>);
 - É utilizado nos campos de texto. Consiste num valor variável de caracteres sendo que o limite é o número de caracteres definido como parâmetro.
- Int;
 - Utilizado nos campos que armazenam valores numéricos.
- TinyInt;
- Char;
- smalldatetime;
 - É utilizado nos campos que armazenam datas.
- smallmoney;
 - É utilizado nos campos que armazenam valores correspondentes a dinheiro.

Após definidos os tipos de campos a utilizar, deu-se a realização do modelo relacional da base de dados, baseando-me no modelo E.A. realizado.

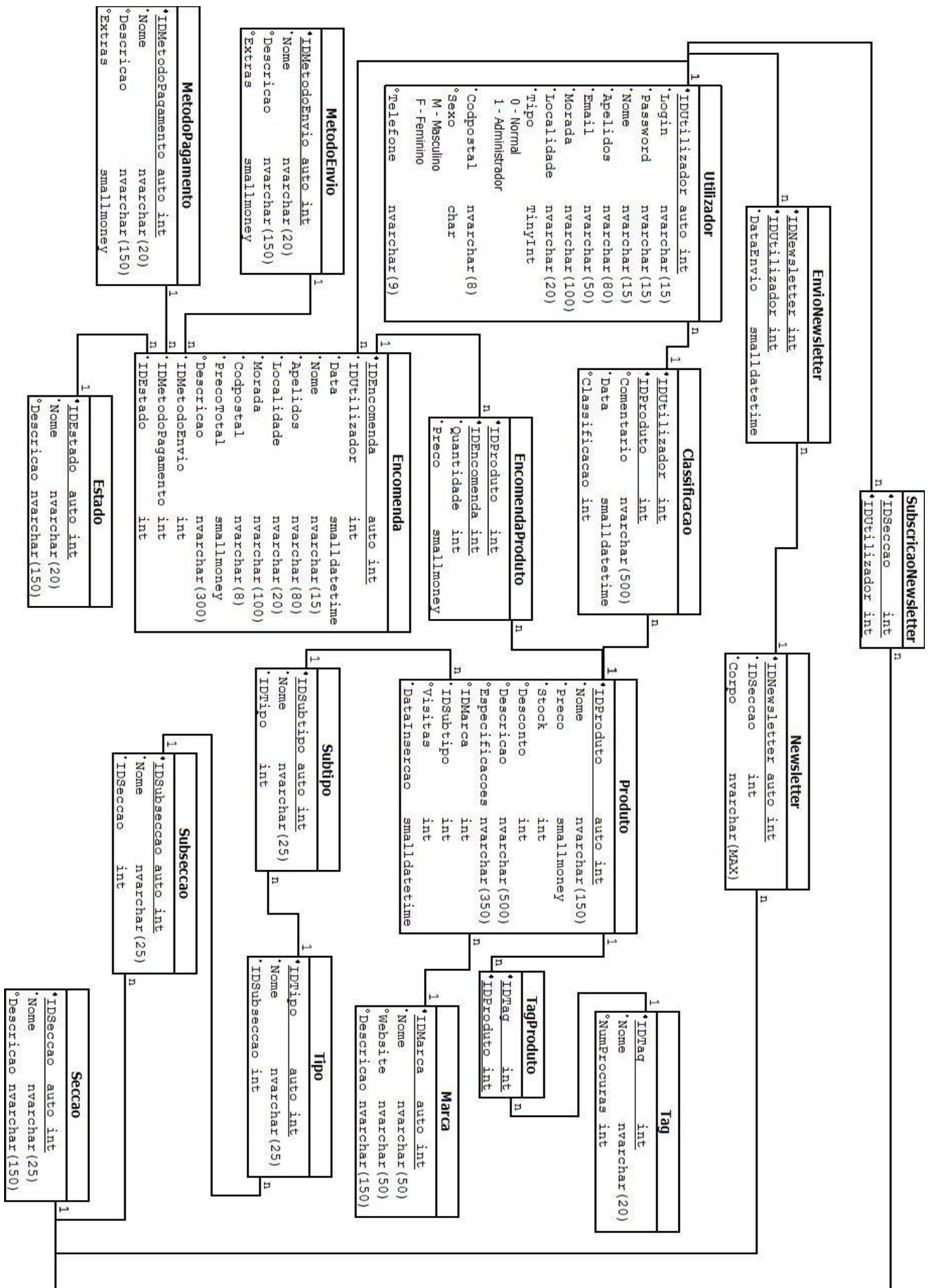


Figura 2 – Modelo relacional da base de dados.

II. Planeamento de algumas funcionalidades

Algumas funcionalidades necessitam de algum planeamento para além da base de dados.

➤ Carrinho de compras

- O carrinho de compras foi implementado utilizando uma estrutura de classes.
- Não valia a pena gastar mais recursos do servidor armazenando o carrinho na base de dados, visto que haveriam demasiados acessos. O carrinho é também temporário e irrelevante para a base de dados enquanto não for efectuada a compra, e nesse caso, o conteúdo da estrutura do carrinho será adicionado na base de dados, na tabela *EncomendaProduto*.

Carrinho
+Produtos: List<Produto> +PrecoTotal: double +NrProdutos: int
+Carrinho() +adicionarProduto(in iProduto:Produto): bool +adicionarQuantidadeProduto(in ID:int): bool +actualizarQuantidades(in Quantidades:List<int>): bool +removerProduto(in ID:int,in ElimQtDs:bool): bool +limparCarrinho(): void -calcularPrecoTotal(): double

Figura 3 – Planeamento da classe Carrinho.

Fase 3: Implementação

Nesta fase, dá-se a implementação, em código, do programa. Esta implementação tem como base e ajuda todo o planeamento efectuado durante a fase de análise.

A fase de implementação está dividida em:

- I. Implementação da interface;
- II. Implementação do Javascript;
- III. Implementação da autenticação;
- IV. Implementação da estrutura de classes;
- V. Implementação das camadas DAL e BLL;
- VI. Implementação de *Web User Controls*;
- VII. Implementação do *Back Office*;

I. Implementação da interface

A implementação da interface foi a fase do projecto que consumiu mais tempo do que o que foi planeado.

Foi desafiante pois decidi desenvolver a interface a partir do zero e para tal houve a necessidade de adquirir certas competências para perceber como são estruturados grande parte dos *websites*.

As cores de um *website* são importantes

Decidi, desde logo, que as cores predominantes do website seriam o branco e o laranja, sendo que as selecções seriam azuis e que haveriam alguns componentes cinzentos.

➤ As CSS (Cascading Style Sheets)

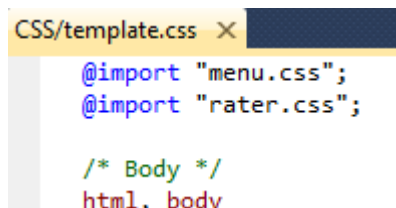
Para desenvolver a interface de um *website* é imprescindível ter conhecimento das CSS e do seu funcionamento.

A estrutura das CSS do meu projecto é a seguinte:

- As CSS dos elementos que aparecem em várias páginas para além do template situam-se nas CSS do template;
- Para todos as CSS de elementos específicos de uma página, estas situam-se no ficheiro .css referente a essa página.

Existe, portanto, aproximadamente um ficheiro .css para cada página, para além do ficheiro .css do *template*. Não havia necessidade de carregar as CSS de elementos que estão em páginas que podem nem chegar a ser carregados. Portanto, a divisão das CSS por página, permite que só seja carregado o que é mesmo necessário.

Em certos casos, como as CSS de determinados elementos importantes (como as CSS do menu principal que se situa no *template*), situam-se também num ficheiro à parte para que seja mais fácil modificar certas coisas durante o desenvolvimento. No entanto, para não aumentar o número de requisitos HTTP fazendo *link* de mais do que um ficheiro .css, estes encontram-se importados pelo ficheiro .css principal (figura).



```
CSS/template.css X
@import "menu.css";
@import "rater.css";

/* Body */
html, body
```

Neste caso, existem 3 ficheiros .css diferentes: "template.css", "menu.css", "rater.css".

O menu principal e o sistema de classificação (*rater*) do website encontram-se no *template*, logo são importados pelo ficheiro template.css. No HTML, em vez de haverem três *links*, correspondentes a cada um dos ficheiros, há apenas um *link*, correspondente ao "template.css":

```
<link href="CSS/template.css" rel="Stylesheet" type="text/css" />
```

➤ Desenvolvimento das imagens

As imagens são outras componentes importantes para a interface de um *website*, trabalhando em conjunto com as CSS. Foram todas desenvolvidas no *software* Adobe Photoshop CS3.

II. Implementação do Javascript

Neste projecto, o javascript tem apenas duas funções: realizar efeitos visuais que tornem a interface mais interessante e fazer o pré-carregamento de algumas imagens importantes.

Estes efeitos visuais em javascript costumam consumir bastante tempo a implementar e testar, portanto decidi usar a API de javascript, **jQuery**.

O ficheiro jQuery.js é importado pela template, pois é utilizado por todas as páginas:

```
<script type="text/javascript" src="Javascript/jquery.js"></script>
```

O jQuery torna muito mais fácil a realização de efeitos visuais, como por exemplo, o efeito do menu principal, em que uma barra azul cobre o fundo laranja:



Figura 4 – Efeitos visuais.

Este efeito é realizado através da 'animação' (durante uma determinada duração, só é vista parte da imagem) de uma imagem que contém parte laranja e parte azul. Essa imagem tem o nome de 'Seccaobg.png'. (figura)



Figura 5 – Imagem de efeito do menu.

Por outras palavras, só é visualizada a parte laranja da imagem e quando o rato passa por cima do menu, essa parte que é visualizada vai sendo, aos poucos, alterada até só conseguirmos visualizar a parte azul.

O código em javascript, na página seguinte, demonstra isso.

```
$( "ul.dropdown li.seccao" ).hover(function () {  
    $(this).css({ backgroundImage: "0 0" })  
    $(this).css("background-image", "url('Imagens/menu/Seccaobg.png')")  
    $(this).stop().animate({ backgroundImage: "(-308 0)" }, { duration: 400 })  
  
    $(this).addClass("hover");  
    $('ul:first', this).css('visibility', 'visible');  
  
}, function () {  
    $(this).css("background-image", "none")  
  
    $(this).removeClass("hover");  
    $('ul:first', this).css('visibility', 'hidden');  
});
```

Isto ocorre no evento 'hover' do menu. Este evento possui duas funções, a função correspondente à passagem do rato por cima do menu (mouseover) e a função correspondente à saída do rato de cima do menu (mouseout).

As duas primeiras linhas da primeira função definem a posição inicial da parte da imagem que é visualizada (0, 0) e define como imagem de fundo a imagem que contém o efeito a realizar (Seccaobg.png).

Em seguida, é realizada a animação, passando por parâmetro o ponto final (-308, 0) e a duração dessa animação.

As duas outras linhas correspondem ao aparecimento do menu secundário.

Na segunda função, quando o rato sai de cima do menu, a imagem de fundo é retirada, deixando apenas o fundo laranja mais uma vez.

A função 'animate' se implementada sem o jQuery seria demasiado complexa e iria consumir bastante tempo a desenvolver.

Tal como já foi referido, o javascript foi implementado apenas para os efeitos visuais e para o pré-carregamento de imagens, portanto não há muito mais a relatar.

III. Implementação da autenticação

Defini que a autenticação do *website* seria possível com ou sem *cookies* e seria implementada baseada em *roles* do utilizador.

De início foi um pouco confuso conjugar as três características (com *cookies*, sem *cookies* e com *roles*), mas após a implementação de uma classe que suportaria todo o processo de autenticação, o objectivo foi cumprido.

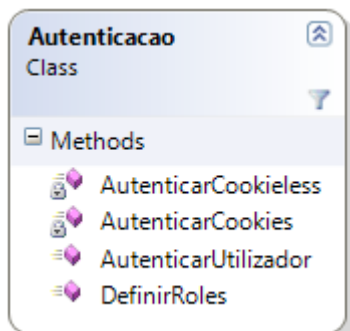


Figura 6 – Classe Autenticação

A classe apresenta quatro métodos:

- AutenticarCookieless();
 - Autentica utilizador se os cookies estiverem desligados.
- AutenticarCookies();
 - Autentica o utilizador usando um cookie;
- AutenticarUtilizador();
 - Verifica se os cookies estão ligados e chama a função correspondente.
- DefinirRoles();
 - Associa os roles do utilizador ao *HttpContext*.

Existem dois roles:

Administrador	Todas as permissões do cliente; Permissões de administração;
Cliente	Permissão de comprar produtos, classificá-los e aceder à página de conta;

Tabela 3 – Tipos de utilizador

O tipo de autenticação, assim como as permissões de acordo com os *roles* são definidos no ficheiro *web.config* da aplicação:


```
<configuration>
  <system.web>
    <!-- (...) -->
    <authentication mode="Forms">
      <forms name="SHOPTEK.ASPXAUTH" loginUrl="Login.aspx"
        defaultUrl="Geral.aspx" protection="All" path="/" timeout="20"
        cookieless="AutoDetect" />
    </authentication>
  </system.web>
```

As permissões às pastas e algumas páginas:

```
<configuration>
  <!-- (...) -->

  <location path="Administração">
    <system.web>
      <authorization>
        <allow roles="Administrador" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>

  <location path="Conta.aspx">
    <system.web>
      <authorization>
        <allow roles="Administrador" />
        <allow roles="Cliente"/>
        <deny users="*" />
      </authorization>
    </system.web>
  </location>

  <location path="Registo.aspx">
    <system.web>
      <authorization>
        <allow users="?" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>

  <location path="Encomendar.aspx">
    <system.web>
      <authorization>
        <allow roles="Administrador" />
        <allow roles="Cliente" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

No acto de autenticação por parte do utilizador, é chamada a seguinte função (quando o login e password inseridos correspondem):

```
Autenticacao.AutenticarUtilizador(retorno.Login, retorno.TipoUtilizador);
```

A função AutenticarUtilizador() e as funções privadas AutenticarCookies() e AutenticarCookieless() da classe Autenticacao:

```
public static void AutenticarUtilizador(string Login, string Roles)
{
    FormsAuthentication.Initialize();

    if (FormsAuthentication.CookiesSupported)
        AutenticarCookies(Login, Roles); //Autentica com cookies
    else
        AutenticarCookieless(Login, Roles); //Autentica sem cookies
}

private static void AutenticarCookies(string Login, string Roles)
{
    FormsAuthenticationTicket tkt;
    string cookiestr;
    HttpCookie ck;

    //Cria um AuthenticationTicket com o 'role' do utilizador actual
    tkt = new FormsAuthenticationTicket(1, Login, DateTime.Now,
        DateTime.Now.AddMinutes(20), false, Roles);

    //Cria o cookie com o Ticket
    cookiestr = FormsAuthentication.Encrypt(tkt);
    ck = new HttpCookie(FormsAuthentication.FormsCookieName, cookiestr);

    ck.Path = FormsAuthentication.FormsCookiePath;

    //Adiciona o cookie criado à lista de cookies
    HttpContext.Current.Response.Cookies.Add(ck);

    //Redirecciona o utilizador
    string strRedirect;
    strRedirect = HttpContext.Current.Request["ReturnUrl"];
    if (strRedirect == null)
        strRedirect = "~/Geral.aspx";
    HttpContext.Current.Response.Redirect(strRedirect, true);
}

private static void AutenticarCookieless(string Login, string Roles)
{
    //Insere na cache o 'role' do utilizador a ser autenticado
    HttpContext.Current.Cache.Insert(Login, Roles, null,
        System.DateTime.UtcNow.AddMinutes(20), System.Web.Caching.Cache.NoSlidingExpiration);

    //Autentica o utilizador
    FormsAuthentication.RedirectFromLoginPage(Login, false);
}
```

Para que a autenticação baseada em *roles* funcione, há necessidade de, a cada requisito de autenticação, estes sejam associados ao utilizador actualmente autenticado:

Global.asax

```
protected void Application_AuthenticateRequest(object sender, EventArgs e)
{
    Autenticacao.DefinirRoles(FormsAuthentication.CookiesSupported);
}
```

A função DefinirRoles() da classe Autenticacao:

```
public static void DefinirRoles(bool cookies)
{
    if (HttpContext.Current.User != null)
    {
        if (HttpContext.Current.User.Identity.AuthenticationType == "Forms")
        {
            System.Security.Principal.IIdentity userId =
                HttpContext.Current.User.Identity;

            if (cookies) //Cookies Ligados?
            {
                string cookieName = FormsAuthentication.FormsCookieName;
                HttpCookie authCookie =
                    HttpContext.Current.Request.Cookies[cookieName];

                FormsAuthenticationTicket authTicket = null;
                authTicket = FormsAuthentication.Decrypt(authCookie.Value);
                string[] roles = authTicket.UserData.Split(new char[] { ',' });

                if (roles != null) //Verifica se existem 'Roles'
                    HttpContext.Current.User = new
                        System.Security.Principal.GenericPrincipal(userId, roles);
            }
            else
            {
                string[] roles = Convert.ToString(
                    HttpContext.Current.Cache[userId.Name]).Split(new char[] { ',' });

                if (roles != null) //Verifica se existem 'Roles'
                    HttpContext.Current.User = new
                        System.Security.Principal.GenericPrincipal(userId, roles);
            }
        }
    }
}
```

IV. Implementação da estrutura de classes

Para além da classe autenticação, houve a necessidade de implementar uma estrutura de classes com o objectivo de organizar a informação relevante do *website* e que esta pudesse ser armazenada nessas classes e transportada por páginas ou parâmetros. Por exemplo, os dados vindos da base de dados em *DataTable* estão menos organizados do que uma estrutura bem definida só com a informação necessária.

Para além disso, defini que o carrinho de compras seria implementado através de uma classe.

➤ Classe Utilizador

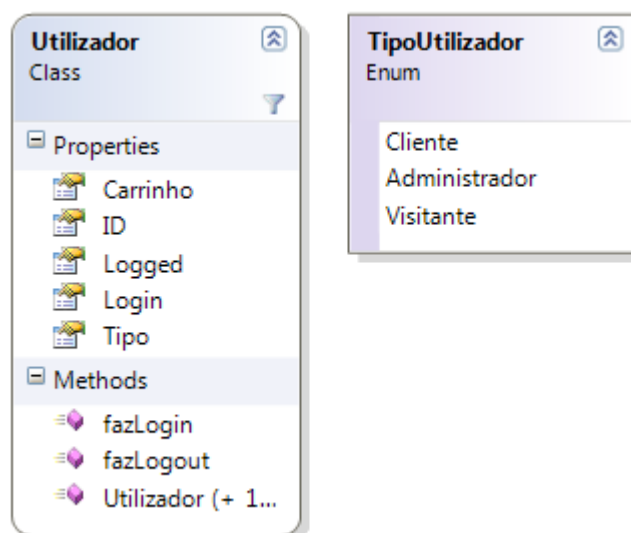
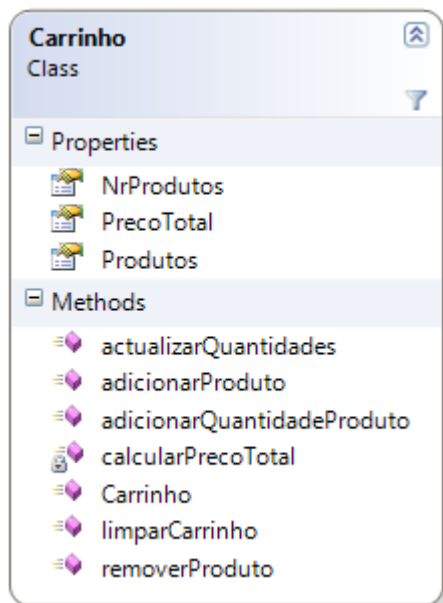


Figura 7 – Classe Utilizador.

A classe utilizador é muito simples e armazena apenas a informação relevante à identificação do utilizador (ID, login, estado e o seu tipo) e o seu carrinho de compras.

Este objecto é mantido nas variáveis de sessão.

➤ Classe Carrinho



A classe carrinho tem como função armazenar produtos que o utilizador introduz no carrinho. Para além de armazenar, é composta por funções que actualizam as quantidades dos produtos, removem produtos, limpam o carrinho, etc.

Existe uma associação de composição entre a classe Carrinho e a classe Utilizador, visto que o Carrinho pertence ao Utilizador e depende da existência deste.

Figura 8 – Classe Carrinho.

Esta classe possui três propriedades:

- NrProdutos;
 - Devolve o número de produtos diferentes existentes no carrinho. (7 exemplares do mesmo artigo contam como 1 produto);
- PrecoTotal;
 - Devolve o método calcularPrecoTotal(), que calcula o preço total dos artigos presentes no carrinho;
- Produtos;
 - Devolve a lista de produtos do carrinho;

Relativamente aos métodos, existem sete:

- actualizarQuantidades();
 - Recebe uma lista de quantidades por parâmetro, cada correspondente a um produto no carrinho e actualiza as quantidades dos produtos;

- adicionarProduto();
 - Recebe um produto por parâmetro e adiciona-o na lista de produtos;
- adicionarQuantidadeProduto();
 - Incrementa a quantidade de um produto existente por um;
- limparCarrinho();
 - Remove todos os produtos do carrinho;
- removerProduto();
 - Remove um produto do carrinho;

```
public bool adicionarQuantidadeProduto(int ID)
{
    for(int c = 0; c < _produtos.Count; c++)
    {
        if (_produtos[c].ID == ID)
        {
            _produtos[c].adicionarQuantidade(1);
            return true;
        }
    }

    return false;
}

public bool adicionarProduto(Produto iProduto)
{
    for(int c = 0; c < _produtos.Count; c++)
    {
        if (_produtos[c].ID == iProduto.ID)
        {
            _produtos[c].adicionarQuantidade(iProduto.Quantidade);
            return true;
        }
    }

    _produtos.Add(iProduto);
    return true;
}

public double PrecoTotal
{
    get{ return calcularPrecoTotal(); }
}
```

```
private double calcularPrecoTotal()
{
    double soma = 0;

    for (int c = 0; c < _produtos.Count; ++c)
        soma += _produtos[c].calcularPrecoTotal();

    return soma;
}

public bool actualizarQuantidades(List<int> Quantidades)
{
    List<int> remover = new List<int>();

    for (int c = 0; c < _produtos.Count; c++)
    {
        if (Quantidades[c] == 0)
            remover.Add(c);
        else
            _produtos[c].alterarQuantidade(Quantidades[c]);
    }

    for(int c = 0; c < remover.Count; c++)
        _produtos.RemoveAt(remover[c] - c);

    return true;
}

public bool removerProduto(int ID, bool ElimQtds)
{
    for (int c = 0; c < _produtos.Count; c++)
    {
        if (_produtos[c].ID == ID)
        {
            if (_produtos[c].Quantidade == 1 || (ElimQtds))
                _produtos.RemoveAt(c);
            else
                _produtos[c].diminuirQuantidade(1);

            return true;
        }
    }

    return false;
}
```

➤ Classe Produto

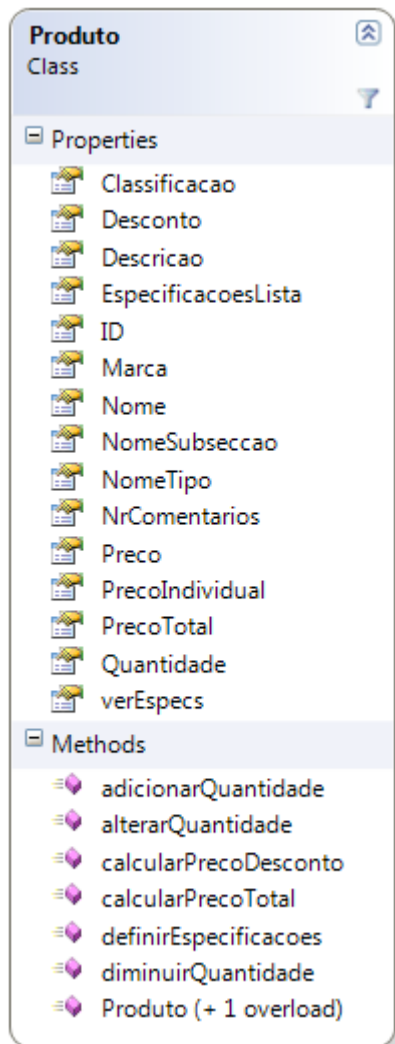


Figura 9 – Classe Produto.

A classe produto armazena os dados de um produto na loja.

Não há muito a relatar acerca desta classe, sendo que as propriedades 'Preco', 'PrecoIndividual' e 'PrecoTotal' devolvem os preços tendo em conta o seguinte:

Preco: Preço de uma unidade de produto sem desconto aplicado;

PrecoIndividual: Preço de uma unidade de produto com o desconto aplicado;

PrecoTotal: Preço total das quantidades de produto com o desconto aplicado;

V. Implementação da DAL e BLL

Para aceder à base de dados de uma forma organizada, dividi o processo de acesso por camadas.

A camada DAL (Data Access Layer) é a camada responsável por fazer a ligação à base de dados, executar o comando SQL e carregar os dados.

A camada BLL (Business Logic Layer) é a intermediária da camada de aplicação e da camada DAL. A sua função é atribuir os parâmetros e comandos SQL a serem executados pela DAL, assim como devolver à camada de aplicação os dados recolhidos de forma organizada.

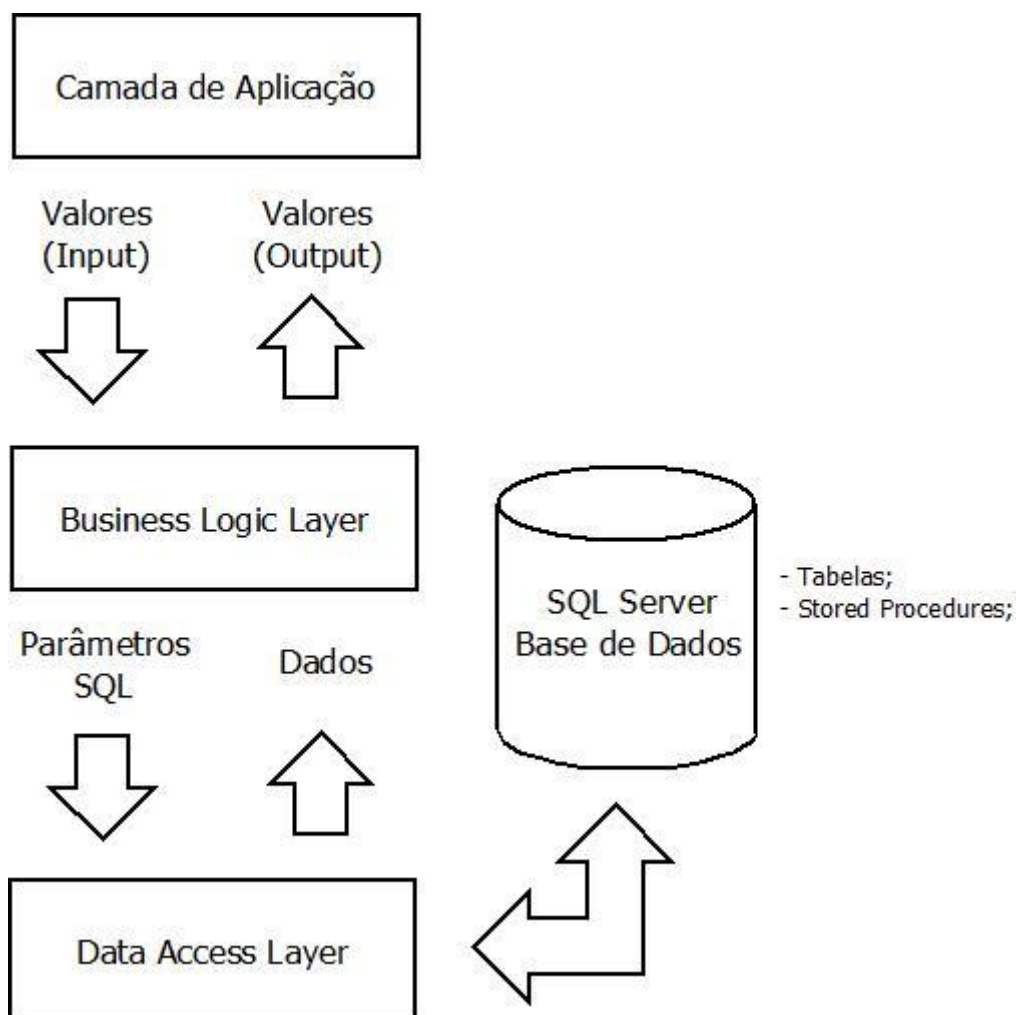


Figura 10 – Camadas de acesso a dados.

➤ Data Access Layer

Cada camada apresenta uma classe que engloba todas as estruturas e funções necessárias para o seu funcionamento.

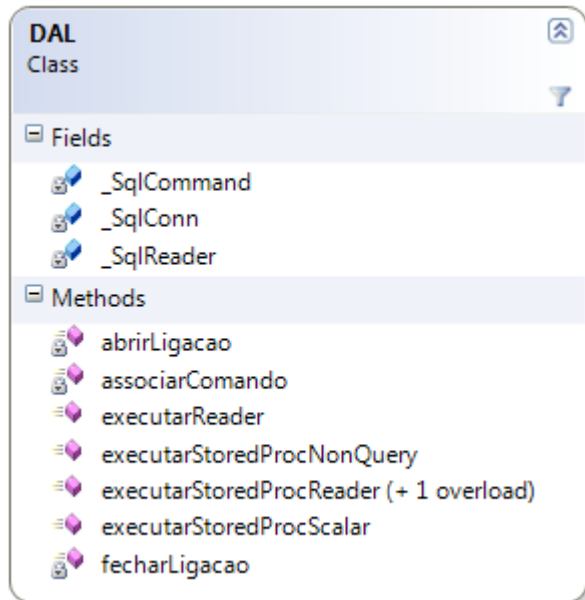


Figura 11 – Data Access Layer.

A classe DAL apresenta 3 métodos privados: `abrirLigacao()`, `associarComando()` e a função `fecharLigacao()` que têm o objectivo correspondente ao seu nome.

Em relação aos outros 4 métodos, estes têm o objectivo de executar um comando SQL ou Stored Procedure e receber os resultados dessa execução:

- `executarReader()`
 - Executa um comando SQL disponibilizado por parâmetro e devolve uma tabela com os resultados.

```
public DataTable executarReader(String sqlCommand, List<SqlParameter> SqlParams){
    DataTable returnTable = new DataTable("returnTable");

    associarComando(sqlCmd);

    for(int c = 0; c < SqlParams.Count; c++)
        _SqlCommand.Parameters.Add(SqlParams[c]);

    abrirLigacao();

    if (_SqlConnection.State == ConnectionState.Open){
        _SqlReader = _SqlCommand.ExecuteReader();
        returnTable.Load(_SqlReader);

        _SqlReader.Close();
        fecharLigacao();
    }

    return returnTable;
}
```

- executarStoredProcNonQuery();
 - Executa uma Stored Procedure disponibilizada por parâmetro e devolve um inteiro com o número de linhas afectadas.

```
public int executarStoredProcNonQuery(String sqlCommand, SqlParameter[] sqlParams)
{
    int retorno = -1;

    associarComando(sqlCmd);

    _SqlCommand.CommandType = CommandType.StoredProcedure;
    _SqlCommand.Parameters.AddRange(sqlParams);

    abrirLigacao();

    if(_SqlConnection.State == ConnectionState.Open){
        retorno = _SqlCommand.ExecuteNonQuery();
    }

    fecharLigacao();

    return retorno;
}
```

- executarStoredProcScalar();
 - Executa uma Stored Procedure e devolve um objecto com o primeiro tuplo da primeira linha de resultados.

```
public object executarStoredProcScalar(String sqlCommand, SqlParameter[] sqlParams)
{
    object resultado = null;

    associarComando(sqlCmd);

    _SqlCommand.CommandType = CommandType.StoredProcedure;
    _SqlCommand.Parameters.AddRange(sqlParams);

    abrirLigacao();

    if (_SqlConnection.State == ConnectionState.Open)
    {
        resultado = _SqlCommand.ExecuteScalar();
    }

    fecharLigacao();

    return resultado;
}
```

- executarStoredProcReader();
 - Executa uma Stored Procedure disponibilizada por parâmetro e devolve uma tabela com os resultados.

```
public DataTable executarStoredProcReader(String sqlCommand, SqlParameter[] sqlParams)
{
    DataTable returnTable = new DataTable("returnTable");

    associarComando(sqlCmd);

    _SqlCommand.CommandType = CommandType.StoredProcedure;

    if(sqlParams != null)
        _SqlCommand.Parameters.AddRange(sqlParams);

    abrirLigacao();

    if (_SqlConnection.State == ConnectionState.Open)
    {
        _SqlReader = _SqlCommand.ExecuteReader();

        returnTable.Load(_SqlReader);

        _SqlReader.Close();
    }

    fecharLigacao();

    return returnTable;
}

//Stored Procedure sem parâmetros
public DataTable executarStoredProcReader(String sqlCommand)
{
    return this.executarStoredProcReader(sqlCmd, null);
}
```

➤ Business Logic Layer

00000

A Business Logic Layer é a maior estrutura de toda a aplicação e por isso não a abordarei com pormenor. Ainda assim, abordarei a estrutura da classe e a estrutura comum de todos os métodos da mesma.

A classe BLL apresenta três grandes classes: bliEncomenda, bliProduto, bliUtilizador.

Dentro dessas classes principais existem outras classes que se relacionam com a classe principal. Por exemplo, um método de envio faz parte de uma encomenda e não de um utilizador. Logo, a classe que trata dos métodos de envio encontra-se na classe bliEncomenda.

Dentro das classes existem métodos que permitem efectuar várias consultas à base de dados.

Os métodos presentes nestas classes apresentam todos uma estrutura comum. A sua função é receber valores por parâmetro, construir parâmetros SQL com esses valores e executar a consulta na base de dados, devolvendo dados de uma forma organizada.

Figura 12 – Business Logic Layer.

Devolver dados de forma organizada significa devolver os dados vindos da DAL, dentro de um objecto devidamente estruturado.

Por exemplo, quando é feita uma consulta por um produto, a DAL retorna um *DataTable*. A BLL recebe esse *DataTable* e em vez de o retornar, organiza os dados presentes nesse mesmo *DataTable* utilizando uma estrutura adequada:

```
public Produto verProduto(int IDProduto)
{
    DataTable tempTable;
    Produto tempProduto = new Produto();
    SqlParameter[] SqlParam = new SqlParameter[1];

    //Conversão dos parâmetros da função para parâmetros SQL
    SqlParam[0] = new SqlParameter("@IDProduto", SqlDbType.Int);
    SqlParam[0].Value = IDProduto;

    //Receber o DataTable com os resultados da função da DAL
    tempTable = DataAccessLayer.executarStoredProcReader("Produto_verProduto",
                                                         SqlParam);

    //Verifica se houveram dados devolvidos
    if (tempTable.Rows.Count > 0)
    {
        //Constrói o objecto a ser devolvido com os dados devolvidos
        tempProduto = new
            Produto(Convert.ToInt32(tempTable.Rows[0]["IDProduto"]),
                    tempTable.Rows[0]["Nome"].ToString(),

            Convert.ToDouble(tempTable.Rows[0]["Preco"]), 1,
            Convert.ToInt32(tempTable.Rows[0]["Desconto"]));

        tempProduto.Classificacao = carregarRating(IDProduto);
        tempProduto.NomeTipo = tempTable.Rows[0]["NomeTipo"].ToString();
        tempProduto.NomeSubseccao =
            tempTable.Rows[0]["NomeSubseccao"].ToString();

        tempProduto.definirEspecificacoes(tempTable.Rows[0]["Especificacoes"].ToString());
        tempProduto.Descricao = tempTable.Rows[0]["Descricao"].ToString();
        tempProduto.Marca = new
            Marca(Convert.ToInt32(tempTable.Rows[0]["IDMarca"]),
                  tempTable.Rows[0]["NomeMarca"].ToString(),
                  tempTable.Rows[0]["Website"].ToString(), "");
        tempProduto.NrComentarios =
            Convert.ToInt32(tempTable.Rows[0]["NrComentarios"]);
    }
    else
        return null; //Devolve null se não houverem resultados

    //Devolve os dados consultados num objecto e de forma organizada
    return tempProduto;
}
```

A função da página anterior, devidamente comentada, revela a estrutura comum dos métodos da BLL.

1. Recebe por parâmetro os valores necessários à consulta;
2. Constrói parâmetros SQL para os parâmetros requisitados pela consulta;
3. Executa o método mais adequado da DAL para a consulta, utilizando os parâmetros SQL construídos anteriormente;
4. Constrói a estrutura a ser devolvida com os dados vindos da DAL no passo 3 (se estes existirem);
5. Devolve *null* se não houveram resultados da DAL;
6. Devolve a estrutura construída se houveram resultados da DAL;

VI. Implementação de *Web User Controls*

No desenvolvimento de aplicações *web* com ASP.NET, é possível implementar *web controls* desenvolvidos pelos utilizadores, para além dos já conhecidos.

Esta funcionalidade traz vantagens e por isso decidi utilizá-la para implementar determinadas funções do meu projecto.

As vantagens dos *web user controls* são claras:

- Separação do código da página e o código do *control* que engloba toda a funcionalidade;
 - Isto permite encapsular as funções do *control*, ficando tudo mais organizado e para além disso, torna-se mais fácil identificar erros;
- Possibilidade de customizar atributos do novo *control*.
- Possibilidade de generalizar o *control* e reutilizá-lo noutras páginas, apresentando resultados diferentes.

Existem duas maneiras de importar um *web user control*:

- Ao nível da página;
 - Em cada página adicionar uma directiva que importa os *web user controls* utilizados pela mesma;

```
<%@ Register Src="~/WebControls/Menu.ascx" TagName="Menu" TagPrefix="uc1" %>
```

- Ao nível do *web.config*;
 - Adicionar os *web user controls* utilizados em toda a aplicação *web* para que estes sejam imediatamente identificados pelas páginas da aplicação sem a necessidade de introduzir uma directiva em cada página.

```
<controls>  
  <add src="~/WebControls/Menu.ascx" tagName="Menu" tagPrefix="uc1"/>  
</controls>
```

Figura 13 – Adicionar um *web user control* pelo *web.config*.

Um exemplo disso é o *web user control* 'ProdutoDestaque' deste projecto.

Precisava de um *control* que mostrasse apenas um produto baseando-se num de vários factores (ex: 'dos mais vendidos') e estivesse presente em várias páginas. Visto que pretendia basear-me em quatro factores, teria que criar quatro *web user controls* iguais, variando apenas no comportamento do código, o que não faria sentido.

Desenvolvi então um *web user control* que apresenta um produto baseando-se num facto passado por atributo:

```
public TipoDestaque Tipo { set { _tipoDestaque = value; } get { return _tipoDestaque; } }

9 public enum TipoDestaque
10 {
11     Destaque = 0,
12     MaisVisto = 1,
13     TopVenda = 2,
14     Rating = 3
15 }
--

<uc1:ProdutoDestaque ID="ProdutoDestaque" runat="server" Tipo="Destaque" />
```

O atributo define em que factor a *query* se vai basear, carregando um produto da base de dados.



Figura 14 – *Web User Control* 'ProdutoDestaque'.

VII. Implementação do back office e notificações

Não havia necessidade de manter o template do front office para o back office: a necessidade de carregar tantas imagens, entre outros aspectos.

Realizei um template muito simples para o back office (figura)

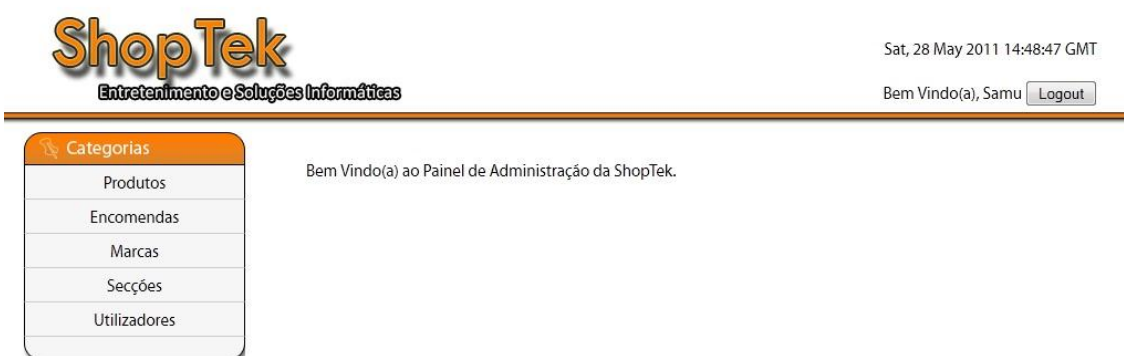


Figura 15 – Template do back office.

De resto, a implementação das funcionalidades de inserção e remoção das mais variadas categorias foi simples mas demorou algum tempo.

Houve, no entanto, uma funcionalidade que gostei especialmente de implementar: o sistema de notificações.

O sistema de notificações, tal como o nome indica, notifica o utilizador do sucesso ou insucesso de operações. Por exemplo: ao inserir um produto, notifica o utilizador de que o produto foi inserido na base de dados com sucesso (figura 16).

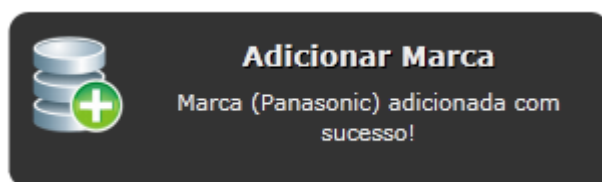


Figura 16 – Notificação de inserção de marca.

Para instanciar uma notificação, criei uma classe que implementa todas as suas características.

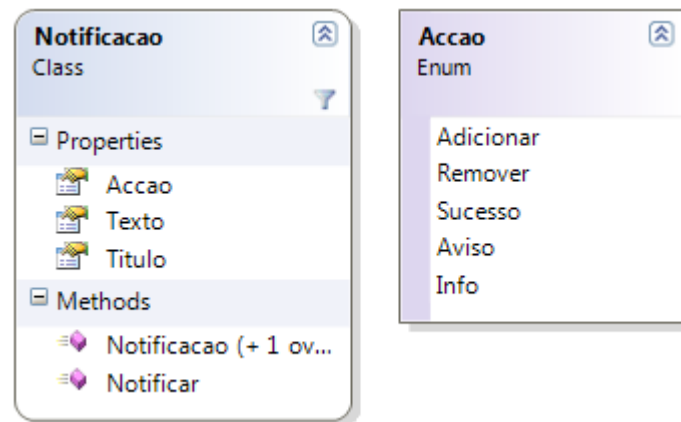


Figura 17 – Classe Notificacao.

```
public void Notificar(Page pagina, Type tipoPagina, string idScript)
{
    string img = "";

    switch (_acao)
    {
        case Accao.Adicionar: img = "plus.png"; break;
        case Accao.Remover: img = "remove.png"; break;
        case Accao.Aviso: img = "warning.png"; break;
        case Accao.Info: img = "info.png"; break;
        case Accao.Sucesso: img = "success.png"; break;
    }

    string script = "<script type='text/javascript' language='javascript'>
        chamarGritter('" + this._titulo + "', '" +
        this._texto + "', '" + img.ToString() + "');
    </script>";

    ScriptManager.RegisterStartupScript(pagina, tipoPagina, idScript,
        script, false);
}
```

A função 'Notificar' coloca a função javascript 'chamarGritter' para ser iniciado no carregamento da página.

Após uma acção, esta função é executada e dá-se um *redirect* para outra página, na qual será mostrada a notificação devido à execução da função 'chamarGritter'.

Versões

No dia 21-04-2011 foi entregue a versão quase completa (98%) da prova de aptidão profissional. Ainda assim, visto que tive tempo, melhorei um pouco para a tornar mais estável.

Versão 1.0

- Entregue a 21-04-2011;
- Todas as funcionalidades explicadas neste relatório;

Versão 1.1

- Entregue a 29-05-2011;
- Correção de vários bugs;
- Implementação de um sistema de notificação no back office;
- Implementação de CSS nos filtros (página Lista.aspx);

Em princípio não haverá necessidade de realizar mais versões até ao dia da apresentação, visto que a 1.1 já se encontra bastante sólida e funcional.

Conclusões

Objectivos realizados

➤ Principais:

- Desenvolver um webiste que implemente uma loja; ✓
 - Essa loja será indicada para a venda de produtos relacionados com a informática, jogos e equipamento audiovisual. ✓
 - Será possível efectuar encomendas através do website; ✓
 - As encomendas podem ser efectuadas após um registo por parte do utilizador, no qual este vai disponibilizar a informação necessária ao processamento da encomenda. ✓
- Os produtos estão organizados por secções do website. Cada secção promove apenas produtos que pertençam a essa mesma secção; ✓
- O *website* será capaz de promover produtos aos utilizadores através de correio electrónico, tendo em conta as áreas mais visitadas por cada utilizador (*Newsletter*); ✗
 - Haverá um tipo de newsletter para cada secção apenas. ✗
 - A subscrição de um utilizador a uma newsletter é facultativa e pode subscrever-se em mais que uma. ✗
- Capacidade de melhorar as procuras dos utilizadores através de vários critérios; ✓
 - Procurar por data (ver artigos mais recentes/antigos), marca, classificação, etc; ✓
- Interacção entre utilizadores através de feedback nos produtos adquiridos; ✓

- A capacidade dos utilizadores registados comentarem e classificarem os produtos; ✓
- As classificações são facultativas para o feedback e o valor da classificação varia de 0 a 5 (inclusive). ✓
- ‘Back office’ simples a fim de gerir o conteúdo da base de dados.
 - Inserir, alterar e remover produtos; ✓
 - Inserir, alterar e remover métodos de pagamento, envio e estados de encomendas; ✓
 - Inserir, alterar e remover marcas; ✓
 - Adicionar/Remover stock aos produtos; ✓
- Secundários:
 - ‘Back office’ que apresente determinadas estatísticas, como por exemplo, as vendas de produtos de informática num determinado mês; ✗
 - Estatísticas por tempo e secção; ✗
 - Melhoramento do sistema de procura através de um sistema de ‘tags’, permitindo encontrar produtos através das mesmas; ✗
 - Recomendar produtos aos utilizadores com base nas últimas visitas/compras efectuadas. ✗

De entre os objectivos planeados, cumpri os principais excluindo o sistema de 'tags', visto que a relevância da funcionalidade não justificava o tempo disponibilizado a realizá-la.

Para além dos objectivos do projecto, fui mais além e realizei a minha interface, o que me deu mais experiência no que toca à visualização e concretização de interfaces.

Apreciação final

A minha apreciação final da prova de aptidão profissional é satisfatória.

Dediquei muito mais tempo ao projecto do que pensei que precisava para o concluir. Houveram fases que, embora não fossem difíceis, consumiam imenso tempo.

Relativamente ao planeamento do projecto, correu razoavelmente bem e o cronograma foi cumprido também razoavelmente.

No que toca ao desenvolvimento, a fase de implementação da interface foi a fase que consumiu mais tempo e, a meio, atrasou ligeiramente o projecto.

Apesar disso, a interface ficou tal e qual como pretendia e reflecte todo o estilo inicialmente pensado

Desenvolver uma prova de aptidão profissional exige aguentar com uma pressão enorme todos os dias e encontrar o máximo de tempo possível durante os dias para dedicar ao desenvolvimento.

Foi difícil, mas os momentos em que as coisas funcionam e começam a ter bom aspecto são os momentos em que mais me apetece trabalhar e foram uma fonte de motivação.

Sinto-me orgulhoso e confiante com o meu projecto e espero obter a nota que tanto ambiciono.