

**ISEL**  
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

# **uOracle**

## SISTEMA DE RECOMENDAÇÕES PERSONALIZADAS

Daniel Ribeiro

Manuel Dias

Rodolfo Morgado

Relatório do projecto realizado no âmbito de Projecto e Seminário do curso de licenciatura em Engenharia  
Informática e de Computadores sob orientação de Miguel Carvalho e Walter Vieira

Setembro de 2009



# Resumo

Este projecto de desenvolvimento, implementado sobre a *Framework .NET*, visa simplificar a criação de aplicações de recomendação personalizada. O sistema proposto, com base na votação atribuída a itens por um utilizador (e.g., filmes, livros), sugere-lhe outros itens. É responsável pelo registo, autenticação e autorização de utilizadores, e pelo registo e catalogação de itens. Integra, de forma fácil, o algoritmo de recomendação que produz o modelo sobre o qual é possível extrair recomendações.



# Agradecimentos

*Aos nossos orientadores, pelo estímulo constante e excelente ambiente de trabalho proporcionado.*

*Ao Professor Miguel Carvalho pela objectividade e pragmatismo com que orientou este projecto.*

*Ao Professor Walter Vieira pela disponibilidade incondicional e acentuado espírito crítico.*

*Ao Professor Artur Ferreira pelo apoio prestado no estudo de algoritmos de recomendação.*

*Aos Professores do Centro de Cálculo do ISEL que contribuíram para este projecto.*

*Aos colegas da disciplina de Projecto e Seminário, pelas ideias e opiniões.*

*Aos nossos familiares, pelo carinho e compreensão nos momentos mais difíceis.*

*A todos vós, o nosso muito obrigado!*



**Daniel**

*Aos meus pais, António e Maria, por tudo o que sou e pela forma como me apoiam e incentivaram durante toda a minha vida.*

*Ao meu irmão Hélio, com quem muito aprendi e pela amizade que sempre tivemos.*

**Manuel**

*Aos meus pais, João e Leonor, pelo amor e carinho que sempre tive e pela motivação que me transmitiram durante o meu percurso académico.*

*Aos meus irmãos, Ana e António, de quem gosto muito.*

*À Gabriela por acreditar em nós.*

**Rodolfo**

*Aos meus pais, Fernando e Alice, por me apoiarem incondicionalmente. Obrigado pelo vosso amor, carinho e dedicação.*

*À Margarida e à Mafalda, que são uma parte de mim.*

*Ao meu irmão Ricardo, pela amizade que nos une.*



# Índice

<b>Resumo</b>	i
<b>Agradecimentos</b>	iii
	v
<b>1 Introdução</b>	1
1.1 Objectivo . . . . .	2
1.2 Arquitectura global do sistema . . . . .	3
1.3 Estrutura do relatório . . . . .	4
<b>2 Formulação do problema</b>	7
2.1 Introdução . . . . .	7
2.2 Características intrínsecas do sistema . . . . .	7
2.2.1 Registo, autenticação e autorização de utilizadores . . . . .	7
2.2.2 Registo e catalogação de itens . . . . .	8
2.2.3 Votação e sugestão de itens . . . . .	8
2.2.4 Integração de algoritmos de recomendação . . . . .	9
2.2.5 Cálculo do modelo, replicação de dados e recomendação . . . . .	9
2.2.6 Manutenção de estado . . . . .	10
2.3 Terminologia . . . . .	10
<b>3 Sistema de recomendações personalizadas</b>	13
3.1 Introdução . . . . .	13
3.2 Análise de requisitos e especificação funcional . . . . .	13

3.2.1	Identificação das entidades participantes no sistema . . . . .	13
3.2.2	Casos de utilização . . . . .	14
3.2.3	Modelo entidade-associação . . . . .	16
3.3	Desenho . . . . .	17
3.3.1	Modelo relacional . . . . .	17
3.3.2	Identificação de classes e suas responsabilidades . . . . .	18
3.4	Implementação . . . . .	30
3.4.1	Introdução . . . . .	30
3.4.2	Organização do código fonte . . . . .	31
3.4.3	Camada de acesso a dados ( <i>DAL - Data Access Layer</i> ) . . . . .	31
3.4.4	<i>UserManager e RoleManager</i> . . . . .	34
3.4.5	<i>ItemManager</i> . . . . .	38
3.4.6	<i>RecommendationManager</i> . . . . .	38
3.4.7	Votação e sugestão de itens . . . . .	39
3.4.8	Replicação . . . . .	41
3.4.9	Paginação . . . . .	42
3.4.10	Testes unitários . . . . .	43
<b>4</b>	<b>Componentes web</b>	<b>45</b>
4.1	Introdução . . . . .	45
4.2	<i>Handlers</i> . . . . .	46
4.3	<i>Custom Controls</i> . . . . .	46
4.3.1	<i>GridViewPagination</i> . . . . .	46
4.3.2	<i>GridViewVotes</i> . . . . .	47
4.3.3	<i>GridViewPredictions</i> . . . . .	48
4.3.4	<i>WebControlItemTypeCategories</i> . . . . .	49
<b>5</b>	<b>Protótipo da aplicação web</b>	<b>51</b>
5.1	Introdução . . . . .	51
5.2	Estrutura . . . . .	51

<b>6 Algoritmo de recomendação</b>	<b>55</b>
6.1 Introdução . . . . .	55
6.2 Implementação . . . . .	56
6.2.1 Contrato com o cliente . . . . .	56
6.2.2 Matriz de interacção . . . . .	56
6.2.3 Matriz de semelhança . . . . .	59
6.2.4 Matriz de predições . . . . .	60
6.2.5 Persistência do modelo de recomendação . . . . .	61
6.2.6 Testes de desempenho . . . . .	62
<b>7 Conclusões</b>	<b>63</b>
7.1 Resultados . . . . .	63
7.1.1 Dificuldades identificadas . . . . .	64
7.2 Considerações para trabalho futuro . . . . .	64
<b>A Documentação auxiliar</b>	<b>67</b>
A.1 Modelo relacional . . . . .	67
A.2 <i>Entity Framework EDM (Entity Data Model)</i> . . . . .	71
A.3 Perspectiva geral do sistema . . . . .	72
A.4 Diagrama de classes do algoritmo de recomendação . . . . .	73



# Listas de Figuras

1.1	Arquitectura global do sistema . . . . .	4
3.1	Modelo Entidade-Associação . . . . .	16
3.2	Exemplo de inserção errónea na associação de uma categoria a um item . . . . .	18
3.3	Diagrama de classes do gestor de utilizadores ( <i>UserManager</i> ) . . . . .	19
3.4	Diagrama de classes da interface <i>IArgs</i> . . . . .	20
3.5	Diagrama de classes da biblioteca do cliente ( <i>ClientLibrary</i> ) . . . . .	21
3.6	Diagrama de classes do gestor de papéis ( <i>RoleManager</i> ) . . . . .	23
3.7	Diagrama de classes do gestor de itens ( <i>ItemManager</i> ) . . . . .	24
3.8	Diagrama de classes do gestor de recomendações ( <i>RecommendationManager</i> ) .	25
3.9	Diagrama de classes do gestor de configurações ( <i>ConfigurationManager</i> ) . .	26
3.10	Diagrama de classes da fábrica de gestores ( <i>ManagerFactory</i> ) . . . . .	27
3.11	Diagrama de classes de <i>ItemSuggestionFactory</i> . . . . .	29
3.12	Representação dos espaço de nomes do sistema . . . . .	31
3.13	Camada de acesso a dados . . . . .	32
3.14	Diagrama de classes da hierarquia <i>UOracleUser</i> . . . . .	33
3.15	Diagrama de classes da implementação de <i>UserManager</i> . . . . .	37
3.16	Diagrama de classes da implementação de <i>RoleManager</i> . . . . .	38
3.17	Diagrama de sequência do processo de recomendação . . . . .	39
3.18	Diagrama de sequência do processo de votação . . . . .	40
3.19	Fluxo de execução do processo de sugestão de itens . . . . .	41
4.1	Diagrama de classes do componente <i>GridViewPagination</i> . . . . .	47
4.2	Diagrama de classes do componente <i>GridViewVotes</i> . . . . .	48

4.3	Diagrama de classes do componente <i>DcGridViewPredictions</i>	49
4.4	Componente <i>WebControlItemTypeCategories</i>	50
5.1	Organização da aplicação <i>web</i>	52
5.2	Organização das <i>master pages</i>	52
6.1	Matrizes envolvidas no cálculo do algoritmo	56
6.2	Matriz de interacção (à esq.) e respectivos índices relativos (à dta.)	57
6.3	Organização da Matriz de Interacção no <i>Array Elem</i>	57
6.4	Pesquisa de uma votação no <i>Array Elem</i>	58
6.5	Representação em vectores do perfil de utilizadores	59
6.6	Cálculo da Matriz de Semelhança	59
6.7	Matriz de Predições antes da normalização	61
6.8	Matriz de Predições após a normalização	61
A.1	<i>Entity Data Model</i> gerado pelo <i>Entity framework</i>	71
A.2	Diagrama de classes do sistema de recomendações	72
A.3	Diagrama de classes do algoritmo de recomendação	73

# **Lista de Tabelas**

2.1	Possível definição de papéis . . . . .	8
2.2	Possível conjunto de pontuações . . . . .	9
3.1	Classes do sistema . . . . .	18
6.1	Testes de desempenho do algoritmo . . . . .	62



# **Lista de Listagens**

3.1	Exemplo de implementação de <i>IArgs</i> . . . . .	22
3.2	Exemplo de personalização de <i>UserManager</i> . . . . .	22
3.3	Configuração de <i>Scores</i> e <i>ManagerFactory</i> . . . . .	28
3.4	Configuração de <i>TrustedHosts</i> e <i>ItemSuggestionFactory</i> . . . . .	30
3.5	Implementação do gatilho <i>TriggerCategories</i> . . . . .	34
3.6	Método <i>CreateUser</i> de <i>MembershipProvider</i> . . . . .	35
3.7	Método <i>CreateUser</i> da classe <i>Membership</i> . . . . .	35
3.8	Método <i>CreateUser</i> da classe <i>AbstractUserManager</i> . . . . .	37
6.1	Estrutura de dados <i>Elem</i> . . . . .	57
6.2	Estrutura de dados <i>Info</i> . . . . .	58



# Capítulo 1

## Introdução

Vivemos numa sociedade de informação. A quantidade e a diversidade de conteúdos disponibilizados, torna o processo de selecção exaustivo e demorado. A Internet é um exemplo desse paradigma social. Os seus conteúdos são concebidos de forma livre e espontânea, podendo ser encontrados através de motores de pesquisa (e.g., *Google*, *Yahoo*). Contudo, os resultados das pesquisas não são personalizados, isto é, não reflectem o gosto pessoal de quem usa o serviço. Desta forma, quem procura, por esse meio, sugestões sobre um produto que pretenda adquirir, vê a sua decisão condicionada, não por falta de informação, mas pela sua disparidade.

Empresas cujos modelos de negócio dependem da interacção com o utilizador através da *Internet*, procuram novas formas de tornar a sua oferta comercial personalizada, esperando assim elevar o grau de satisfação do cliente. A *Amazon* ([www.amazon.com](http://www.amazon.com)) foi um dos pioneiros, oferecendo recomendações com base nos produtos previamente adquiridos pelos clientes. Contudo, a aquisição de um produto, mesmo que por recomendação, não implica satisfação garantida. Este problema constata-se no sistema da *Amazon*, onde após a aquisição do produto X, é sugerido um conjunto Y de produtos, adquiridos por consumidores do produto X. Para solucionar este problema, a *Amazon* solicita aos seus clientes que manifestem a sua preferência relativamente aos produtos adquiridos através de um sistema de votação, construindo dessa forma um perfil que caracteriza os seus gostos pessoais. Com base nesse perfil, a *Amazon* recorre a um algoritmo proprietário de recomendação, mais concretamente *Item to Item Collaborative Filtering* [3], para sugerir produtos que possam interessar aos seus clientes.

O *Collaborative Filtering* é uma das técnicas com maior aceitação na área do comércio electrónico [1, 2]. Tradicionalmente, utiliza como fonte de dados a interacção consumidor/produto, sendo ignorados os atributos que individualmente os caracterizam. Com base nessa interacção são estabelecidas vizinhanças, aproximando consumidores com interacções semelhantes. As recomendações são efectuadas segundo essas vizinhanças, isto é, recomendando produtos adquiridos por consumidores que no passado tiveram interacções semelhantes.

Esta área tem despertado interesse na comunidade científica, destacando-se o trabalho desenvolvido pelo *GroupLens Research Group* ([www.grouplens.com](http://www.grouplens.com)), um grupo de investigação do Departamento de Engenharia e Ciências da Computação da Universidade de Minnesota, que desde 1994 se dedica ao estudo e desenvolvimento de algoritmos de recomendação baseados em *Collaborative Filtering*. Para testar o sistema de recomendações num cenário real, a *GroupLens* desenvolveu o projecto *MovieLens* ([www.movielens.com](http://www.movielens.com)), onde o utilizador constrói um perfil através da votação em filmes, com o objectivo de posteriormente obter recomendações. Neste momento, o projecto *MovieLens* conta com, aproximadamente, 71000 utilizadores, 10600 filmes e 10 milhões de votações efectuadas. Realça-se que a *GroupLens* é um projecto sem objectivos comerciais, e disponibiliza, para fins académicos, o respectivo conjunto de dados.

Existem outros projectos semelhantes ao *MovieLens*, mas com fins comerciais, dos quais se destacam o *uLike* ([www.ulike.com](http://www.ulike.com)), e o muito popular *IMDB* (*Internet Movie Database*) ([www.imdb.com](http://www.imdb.com)). A *eBay Inc.* ([www.ebay.com](http://www.ebay.com)), e a *Apple Inc.* através do *iTunes* ([www.itunes.com](http://www.itunes.com)) também oferecem recomendações nos seus sítios.

O interesse crescente em sistemas de recomendações fomentou o aparecimento de soluções *open source*, tais como o *Taste* (<http://taste.sourceforge.net>), e o *CoFE* (*Collaborative Filtering Engine*) (<http://eecs.oregonstate.edu/iis/CoFE/>), dois motores escritos em Java, implementando algoritmos de recomendação de *Collaborative Filtering*.

Constata-se que as soluções *open source* mencionadas facilitam a integração de uma grande diversidade de algoritmos de recomendação. Contudo, empresas que através deste meio pretendam oferecer recomendações aos seus clientes, necessitam de uma infra-estrutura que conte tempo o registo de utilizadores, a autenticação e autorização dos mesmos, bem como o registo e catalogação de produtos. Assim, é pertinente pensar-se num sistema responsável por facilitar o desenvolvimento da referida infra-estrutura e que, simultaneamente, suporte a integração de algoritmos de recomendação.

## 1.1 Objectivo

Este projecto de desenvolvimento visa a concepção e implementação de uma biblioteca de classes que facilite o desenvolvimento de aplicações de recomendação. O sistema proposto, com base na votação atribuída a **itens** pelo utilizador, sugere-lhe, de forma personalizada, outros itens. Entenda-se que itens podem ser filmes, livros, eventos, entre outros. O sistema responde a duas classes de perguntas de forma personalizada: *quais os itens recomendados; se determinado item é recomendado*.

O sistema deverá possibilitar:

- O registo e autenticação de utilizadores;
- O registo e catalogação de itens;
- O controlo de autorizações (*role-based*<sup>1</sup>) no acesso a recursos por parte dos utilizadores;
- A criação de perfis de utilizadores (i.e., as votações dos utilizadores em itens);
- A integração fácil de algoritmos de recomendação (e.g., *Collaborative Filtering*);
- A realização de recomendações, com base no cálculo efectuado pelo algoritmo de recomendação utilizado.

## 1.2 Arquitectura global do sistema

O sistema é constituído por dois subsistemas autónomos, representados na figura 1.1. Esta divisão da arquitectura em dois subsistemas, com dois *SGBDs*<sup>2</sup> distintos, providencia *escalabilidade*. Consoante o número de utilizadores, o cálculo do *modelo*<sup>3</sup> pode ser efectuado sobre o *SGBD* do subsistema 1 ou, numa situação crítica, em que o número de utilizadores possa comprometer o desempenho do sistema, é possível utilizar um *SGBD* alternativo.

O subsistema 1 é responsável pelo registo, autenticação e autorização de utilizadores, gestão de itens, perfis<sup>4</sup> e recomendações.

O subsistema 2 gera o modelo de forma autónoma, usando o algoritmo de recomendação integrado no sistema.

Este projecto foi implementando com recurso a tecnologias *Microsoft*. Foi utilizada a *Framework .NET 3.5* no desenvolvimento da biblioteca de classes, e a *ADO.NET Entity Framework* na implementação da camada de acesso a dados. Para persistência de dados e integração do algoritmo de recomendação optou-se pelo *SGBD SQL Server, versão 2008*.

Com o objectivo de minimizar o esforço despendido pelo cliente no desenvolvimento de aplicações *web* sobre este sistema, são disponibilizados *Custom Controls .NET* incorporando a lógica necessária para executar algumas das funções chave do sistema (e.g., votação).

---

<sup>1</sup>Autorização baseada em papéis, isto é, nas funções desempenhadas por utilizadores.

<sup>2</sup>Sistema Gestor de Bases de Dados.

<sup>3</sup>Estabelece um valor de *predição* por cada associação utilizador/item.

<sup>4</sup>As votações dos utilizadores em itens.

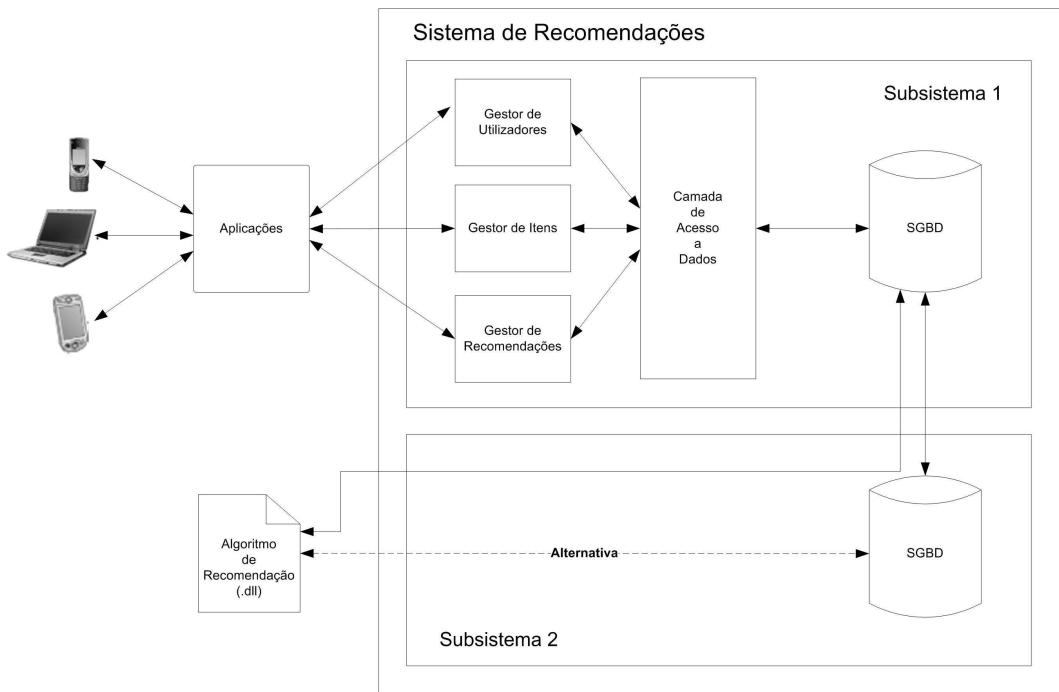


Figura 1.1: Arquitectura global do sistema

### 1.3 Estrutura do relatório

Este relatório está organizado em sete capítulos. O presente capítulo, de cariz introdutório, expõe as motivações deste trabalho, traçando as suas metas e objectivos. Apresenta, de forma sucinta, a solução para o problema. Termina com a descrição do relatório em termos estruturais.

O capítulo 2, destina-se à caracterização do problema e respectiva contextualização. São identificados os principais problemas na concepção do sistema de recomendações e apresentadas algumas soluções generalistas. É efectuada uma descrição detalhada das suas componentes mais relevantes e estabelecida a terminologia necessária ao seu entendimento.

O capítulo 3, documenta a concepção do sistema de recomendações *UOracle*, seguindo uma cronologia organizada em três fases. A fase de análise de requisitos funcionais, onde se apresenta um conjunto de casos de utilização do sistema e se esboça o modelo de dados. A fase de desenho, onde se define o conjunto de classes que representam o problema a tratar. A fase de implementação, onde são concretizados e testados os modelos conceptuais propostos.

O capítulo 4, apresenta os componentes concebidos com o intuito de minimizar o esforço do **cliente**<sup>5</sup> no desenvolvimento de aplicações *web*.

O capítulo 5, retrata os aspectos estruturais do protótipo (aplicação *web*) desenvolvido para testar as capacidades de integração do sistema de recomendações.

<sup>5</sup>Entidade que utiliza o sistema para criar aplicações.

O capítulo 6, ilustra o desenvolvimento do algoritmo de recomendação, identificando os problemas inerentes à sua concretização e a forma como estes foram ultrapassados.

O capítulo 7, apresenta as conclusões gerais sobre o trabalho desenvolvido e um conjunto de considerações para trabalho futuro.

Em apêndice, são incluídos pormenores de implementação, relevantes para um entendimento mais profundo do funcionamento interno do sistema.



# Capítulo 2

## Formulação do problema

### 2.1 Introdução

Um sistema de recomendações personalizadas depende da contribuição dos seus utilizadores, quer esta seja implícita, isto é, extraída a partir de acções por estes realizadas (e.g., a compra de um produto); ou explícita, no caso dos utilizadores participarem activamente no sistema através de apreciações, por exemplo, sobre produtos que tenham adquirido. O sistema proposto baseia-se na contribuição explícita de utilizadores através do conjunto das suas votações em itens, ao qual se atribui a designação de **perfil**.

No presente capítulo, são identificados os principais constrangimentos na concepção de um sistema com as características propostas. Para concluir, é estabelecida a terminologia necessária ao seu entendimento.

### 2.2 Características intrínsecas do sistema

#### 2.2.1 Registo, autenticação e autorização de utilizadores

Para que o utilizador vote em itens, construindo dessa forma um perfil que reflecta os seus gostos pessoais, é fundamental que o sistema controle o acesso a essa informação através de um processo de autenticação e autorização. Assim, a criação de um perfil implica um registo prévio onde o utilizador fornece a informação necessária para que possa ser identificado inequivocavelmente pelo sistema (e.g., o endereço *e-mail* e uma senha de acesso definida pelo próprio). Importa salientar que, no processo de registo, o cliente é responsável por estabelecer um canal seguro de comunicação (e.g., *SSL - Secure Socket Layer*) para que a informação confidencial (e.g., senha de acesso) não seja transportada em claro.

Num sistema com estas características, prevê-se que o cliente delegue aos seus funcionários tarefas de administração e moderação. Para que tal seja possível, o sistema deve contemplar um esquema configurável de autorização, dando autonomia ao cliente na definição de papéis que permitam controlar o acesso dos utilizadores, restringindo as suas acções mediante as funções que estes desempenhem. Na tabela 2.1 ilustra-se uma possível organização por parte de um cliente.

Papel	Ações
Administrador	Criar tipos e categorias, criar moderadores
Moderador	Criar itens
Utilizador Autenticado	Votar, obter recomendações

Tabela 2.1: Possível definição de papéis

## 2.2.2 Registo e catalogação de itens

O título de um item não o identifica inequivocamente. Podemos constatar esta situação quando existem duas ou mais versões de uma mesma obra (e.g., *War of the Worlds*<sup>1</sup>). Assim, é necessário criar uma propriedade extrínseca ao item que constituirá o seu identificador único.

Para facilitar o processo de pesquisa e oferecer recomendações mais específicas, um item deve ter um tipo associado (e.g., cinema) e, facultativamente, ser inserido em uma ou mais categorias (e.g., acção, drama), tendo estas de estar associadas ao tipo escolhido. Esta organização garante flexibilidade ao sistema, podendo os itens e categorias ser estendidos e associados entre si numa relação de *M:N*.

## 2.2.3 Votação e sugestão de itens

A construção de perfis reflectindo os gostos pessoais dos utilizadores é conseguida à custa da votação em itens.

O processo de votação é constituído por duas fases: a fase preparatória, em que o utilizador pesquisa o item sobre o qual pretende manifestar a sua opinião; a fase de votação, em que o utilizador efectiva o voto, atribuindo uma pontuação ao item.

O valor da pontuação atribuída e a sua designação deve ser configurável, podendo o cliente definir um conjunto de pontuações a atribuir a itens no processo de votação. A tabela 2.2 mostra uma possível configuração.

---

<sup>1</sup>A guerra dos mundos - obra literária de H.G. Wells com duas adaptações cinematográficas (1953 e 2005).

Designação	Valor
Não Votado	0
Péssimo	1
Mau	2
Razoável	3
Bom	4
Excelente	5

Tabela 2.2: Possível conjunto de pontuações

Pode dar-se o caso de o item sobre o qual o utilizador pretende manifestar a sua opinião ainda não constar na base de dados. Assim, importa facultar um meio para o utilizador sugerir itens. Esta abertura tem como principal vantagem contribuir para a dinâmica do sistema. Contudo, é propícia a erros, tanto na atribuição do título, como no tipo seleccionado pelo utilizador. Por esse motivo, os itens sugeridos apenas devem ser inseridos após validação por parte de um moderador.

#### 2.2.4 Integração de algoritmos de recomendação

Pressupõe-se que o sistema possa ser acedido por um elevado número de utilizadores. Assim, é imprescindível oferecer capacidade de resposta, tanto na interacção com os utilizadores, como na eficiência do cálculo do modelo.

Para que a integração do algoritmo de recomendação seja eficiente, é primordial que este seja executado na máquina onde se encontram os dados de perfil dos utilizadores. Estando a sua persistência a ser efectuada numa base de dados relacional, importa realçar que existem neste momento tecnologias preparadas para executar código *managed* no *SGBD*.

Desde a versão 2005, o *SQL Server* suporta a integração do *CLR* [9, 10, 11], sendo possível executar código *managed* no mesmo processo que o *SQL Server*. Esta partilha do processo traduz-se em eficiência, sendo o acesso a recursos (e.g., memória, tempo de *CPU*) controlado pelo *SQL Server*. Comparativamente com a linguagem *T-SQL*, para além de uma maior facilidade de programação, o algoritmo de recomendação pode ser integrado no sistema com níveis de desempenho mais elevados.

#### 2.2.5 Cálculo do modelo, replicação de dados e recomendação

Sendo o cálculo do modelo de recomendação uma tarefa computacionalmente exigente, importa garantir que essa acção não prejudica a interacção com o utilizador no acesso aos dados de

perfil. Uma solução possível para este problema consiste na replicação dessa informação para que possa ser utilizada exclusivamente pelo algoritmo no cálculo do modelo.

No sistema deve ser equacionado que em algumas circunstâncias não será possível disponibilizar uma recomendação ao utilizador, quer por este não ter ainda votado em itens, ou pelo facto do registo do utilizador ser recente e o seu perfil não ter sido considerado no cálculo do modelo. Assim, devem ser idealizadas soluções alternativas, tais como um *top* de recomendações não personalizadas (recomendando os itens mais votados), para que o utilizador não veja a sua expectativa totalmente gorada.

### 2.2.6 Manutenção de estado

Estando o sistema vocacionado para a construção de aplicações *web*, e sendo o *HTTP (Hyper Text Transfer Protocol)* um protocolo *stateless*<sup>2</sup>, cada pedido efectuado ao servidor implica a verificação das credenciais do utilizador, o que em circunstâncias normais se traduz numa consulta à base de dados. Considerando esta penalização de desempenho, o cliente deve contemplar a manutenção de estado através de um mecanismo auxiliar (e.g., *cookies*) para minimizar as ligações à base de dados efectuadas pelo sistema no processo de autenticação.

## 2.3 Terminologia

Os termos utilizados para caracterizar cada uma das entidades participantes, apenas fazem sentido devidamente contextualizados. Apresenta-se, de forma sumária, o seu significado no contexto actual, bem como as suas propriedades mais importantes.

**Cliente** - Quem utiliza o sistema de recomendações para criar aplicações;

**Utilizador** - Quem utiliza a aplicação desenvolvida pelo cliente;

**Papel** - Representa uma classe de utilizadores (e.g., Administrador, Moderador, Utilizador Autenticado), com autorização para realizar um determinado número de funções;

**Item** - Representa um objecto do domínio de aplicação (e.g., um filme, um livro), sobre o qual os utilizadores efectuam votações e obtêm recomendações;

**Pontuação**. Grandeza escalar que quantifica a satisfação do utilizador relativamente ao *item*;

**Tipo** - Área temática onde se insere o item (e.g., cinema, teatro);

**Categoria** - A categoria em que se insere o item (e.g., drama, romance). Dois itens podem ter tipos diferentes e, mesmo assim, partilharem a mesma categoria, isto é, um item pode ser do tipo *Cinema* e outro do tipo *Teatro*, mas ambos se enquadrem na categoria *Drama*.

---

<sup>2</sup>Protocolo sem estado, isto é, não prevê a persistência de informação entre pedidos subsequentes.

**Votação** - Associação utilizador/item, a que se atribui uma pontuação;

**Perfil** - Conjunto de votações, que define o gosto pessoal do utilizador;

**Predição** - Um valor calculado pelo algoritmo de recomendação relativamente à associação utilizador/item e que mede a expectativa dessa associação;

**Modelo** - Estrutura de dados calculada pelo algoritmo de recomendação que estabelece um valor de predição por cada associação utilizador/item;

**Recomendação** - Acto de recomendar, baseado no modelo.



# **Capítulo 3**

## **Sistema de recomendações personalizadas**

### **3.1 Introdução**

Este capítulo retrata as três fases de desenvolvimento da biblioteca de classes do sistema de recomendações *UOracle*.

Na fase de análise de requisitos e especificação funcional, são identificadas as entidades participantes com o objectivo de clarificar o seu significado. O seu contributo para o aspecto funcional do sistema é reconhecido através de *casos de utilização*. É definido o modelo entidade-associação, a base para o modelo relacional de dados.

Na fase de desenho, são definidas as classes que representam o sistema, sendo identificadas as suas responsabilidades e a forma como se relacionam entre si.

Na fase de implementação, são concretizados os modelos conceptuais propostos na fase de desenho.

### **3.2 Análise de requisitos e especificação funcional**

O sistema é responsável pelo registo, autenticação e autorização dos utilizadores, e pelo registo e catalogação de itens. Integra o algoritmo de recomendação que produz o modelo para dar resposta às solicitações de recomendação efectuadas pelos utilizadores, com base nos seus perfis.

#### **3.2.1 Identificação das entidades participantes no sistema**

Para minimizar redundância na descrição textual, a presente secção apenas caracteriza os aspectos funcionais das entidades referidas em 2.3.

**Utilizador** - Personifica o utilizador. Estabelece um perfil através da votação em itens. Solicita *recomendações*.

**Papel** - Utilizado no processo de autorização para que o acesso a recursos seja atribuído mediante a função desempenhada pelo utilizador no sistema de recomendações (e.g., Administrador, Moderador, Utilizador Autenticado);

**Item** - O item a registar e catalogar;

**Pontuação** - Grau de satisfação do utilizador relativamente ao item;

**Tipo** - O item é caracterizado por um e só um tipo (e.g., cinema, teatro);

**Categoria** - O item está associado a um conjunto de categorias. Contudo, este deve ser um subconjunto das categorias definidas para o tipo do item;

**Votação** - O utilizador vota no item, atribuindo-lhe uma pontuação;

**Predição** - O valor que mede a expectativa relativamente à associação utilizador/item para um determinado utilizador.

### 3.2.2 Casos de utilização

#### Registo de utilizador.

1. O utilizador regista-se, disponibilizando o seu endereço de *e-mail* e uma senha de acesso;
2. É enviada uma mensagem para o *e-mail* disponibilizado pelo utilizador, contendo o endereço de activação;
3. O utilizador activa o seu registo através do endereço de activação.

#### Autenticação e autorização.

1. O utilizador insere o seu endereço de *e-mail* e a sua senha de acesso, através de uma das interfaces disponibilizadas (e.g., aplicação *web*);
2. Caso as credenciais sejam válidas, o utilizador é autenticado e autorizado a aceder ao sistema. Caso contrário, é redireccionado para a área de registo.

#### Votação e sugestão de itens.

1. O utilizador efectua uma pesquisa sobre o item em que pretende votar;
2. No caso do item existir, o utilizador atribui-lhe uma pontuação. Caso contrário, pode sugerir o item, para que seja analisado por um moderador e, posteriormente, adicionado ao sistema.

**Recomendações (caso 1).** Quais os itens recomendados para o utilizador.

1. O utilizador selecciona um tipo e solicita uma recomendação;
2. O sistema devolve uma lista de itens recomendados bem como o seu grau de recomendação.

**Recomendações (caso 2).** Se determinado item é recomendado ao utilizador.

1. O utilizador selecciona o item a avaliar.
2. O sistema devolve uma predição.

### **Inserção de itens.**

1. O utilizador introduz as suas credenciais;
2. Caso tenha privilégios suficientes (e.g., moderação), tem acesso à área de inserção de itens. Caso contrário, é redireccionado para a área de *login*;
3. O utilizador insere um título e selecciona o tipo que pretende associar ao item. Opcionalmente pode definir o conjunto de categorias a associar ao item.

### **Validação das sugestões de itens.**

1. O utilizador introduz as suas credenciais;
2. Caso tenha privilégios suficientes (e.g., moderação), tem acesso à área de validação de sugestões. Caso contrário, é redireccionado para a área de *login*;
3. Consulta uma lista de sugestões de itens submetidos por utilizadores;
4. Caso se confirme a pertinência da sugestão, esta é concretizada num novo item. Caso pretenda, o utilizador pode associar um conjunto de categorias ao item.

### **Definição de tipos e categorias.**

1. O utilizador introduz as suas credenciais;
2. Caso tenha privilégios suficientes (e.g., administração), tem acesso à área de definição de tipos e categorias. Caso contrário, é redireccionado para a área de *login*;
3. O utilizador cria tipos e categorias a associar a itens;
4. Estabelece associações entre categorias e itens.

### 3.2.3 Modelo entidade-associação

A identificação das entidades participantes efectuada em 3.2.1, constitui a base para modelo entidade-associação representado na figura 3.1.

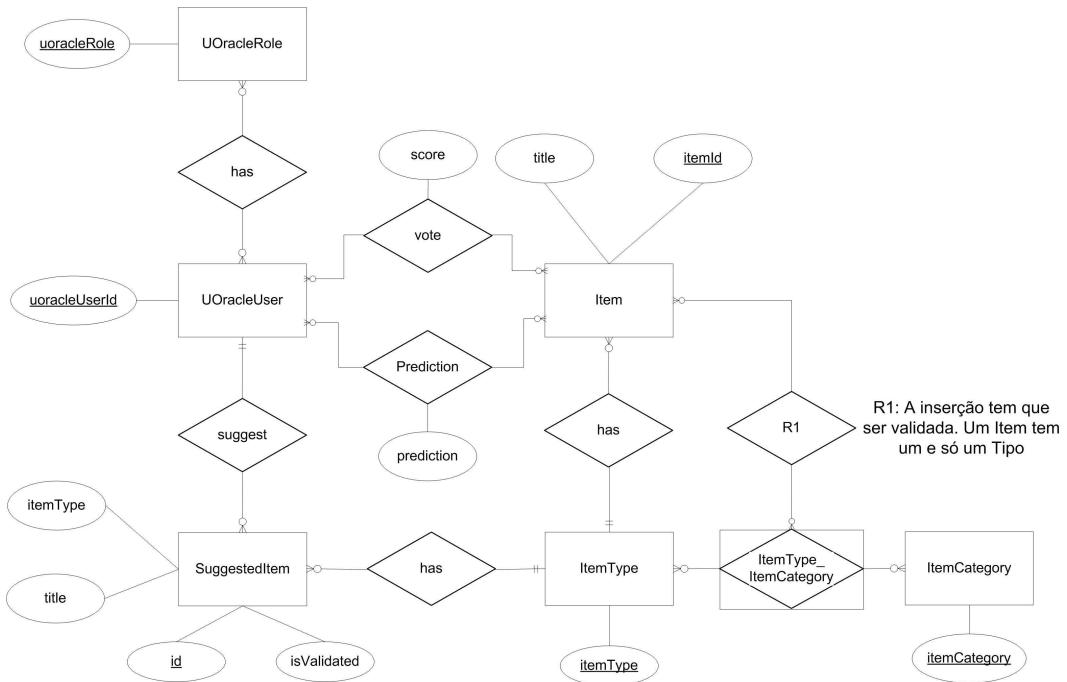


Figura 3.1: Modelo Entidade-Associação

## 3.3 Desenho

### 3.3.1 Modelo relacional

**Análise de obrigatoriedade, esquemas de relação, e chaves.** Tendo como base o modelo entidade-associação, é aplicado um conjunto de regras de transformação [5] para construir o modelo relacional presente no apêndice A.

**Normalização.** O modelo lógico extraído do modelo conceptual, presente no apêndice A, apresenta todos os esquemas de relação normalizados até à terceira forma normal, preservando as dependências funcionais.

**Restrições.** O modelo relacional não é suficiente para impor todas as restrições propostas relativamente à catalogação de itens. Embora o modelo restrinja as categorias associadas a um item através de uma chave estrangeira para a entidade associativa *ItemType\_ItemCategory*, não há garantia que esta chave aponte para uma associação cujo tipo corresponda ao definido para o item. Um exemplo de uma inserção errónea, não detectada pelas restrições imposta pelo modelo relacional seria, por exemplo, um item do tipo *Cinema* estar associado a um registo de *ItemType\_ItemCategory* do tipo *Teatro*. Esta restrição terá de ser imposta no momento de inserção e actualização de dados, por meio de acções controladas por procedimentos armazenados ou por gatilhos.

Na imagem 3.2 está ilustrada uma possível inserção errónea na associação de uma categoria a um item. Neste exemplo, o tipo *Teatro* não corresponde ao tipo do item, mas as restrições do modelo relacional não são suficientes para evitar a inserção.

**Item**

itemId	Title	ItemType
1	Blade Runner	Cinema
2	Memento	Cinema

**ItemType\_ItemCategory**

ItemType	ItemCategory
Teatro	Drama
Cinema	Acção

Inserção errónea.  
Teatro difere do tipo do item 1.

**Item\_ItemTypeItemCategory**

itemId	itemType	itemCategory
1	Teatro	Drama
1	Cinema	Acção

X  
✓

Figura 3.2: Exemplo de inserção errónea na associação de uma categoria a um item

### 3.3.2 Identificação de classes e suas responsabilidades

As entidades identificadas em 3.2.1 podem ser traduzidas directamente nas classes apresentadas na tabela 3.1. Para cada uma das classes é definida uma interface expondo os métodos públicos. A designação das interfaces segue a notação *Pascal* [12], sendo composta pelo nome da classe prefixado com a letra *I* (e.g., *IUOracleUser*). Uma vez que documentação do código fonte contém a descrição exaustiva dos métodos, apenas são mencionados neste relatório os aspectos relevantes para a compreensão do sistema.

Entidade	Classe
Utilizador	<i>IUOracleUser</i>
Papel	<i>IUOracleRole</i>
Item	<i>item</i>
Pontuação	<i>Score</i>
Tipo	<i>ItemType</i>
Categoria	<i>ItemCategory</i>
Votação	<i>Vote</i>
Predição	<i>Prediction</i>

Tabela 3.1: Classes do sistema

A lógica de negócio que se perspectiva ser comum para a maioria das aplicações é definida

nas hierarquias de classes: gestor de utilizadores (*UserManager*), gestor de papéis (*RoleManager*), gestor de itens (*ItemManager*) e gestor de recomendações (*RecommendationManager*). A configuração do sistema, que possibilita ao cliente definir elementos que caracterizam o seu modelo de negócio, é da responsabilidade do gestor de configurações (*ConfigurationManager*).

### Gestor de utilizadores (*UserManager*)

Responsável pela gestão da informação relativa ao utilizador, onde se enquadra o registo e o perfil, bem como o respectivo processo de autenticação.

A hierarquia de classes *UserManager*, representada na figura 3.3, é constituída pela interface *IUserManager*, a classe abstracta *AbstractUserManager* e a classe concreta *UserManager*. O cliente tem três opções à sua disposição na gestão de utilizadores: utilizar a implementação *UserManager*, delegando a responsabilidade ao sistema; utilizar *AbstractUserManager*, caso pretenda personalizar o gestor; utilizar *IUserManager*, caso prefira realizar uma implementação de raiz.

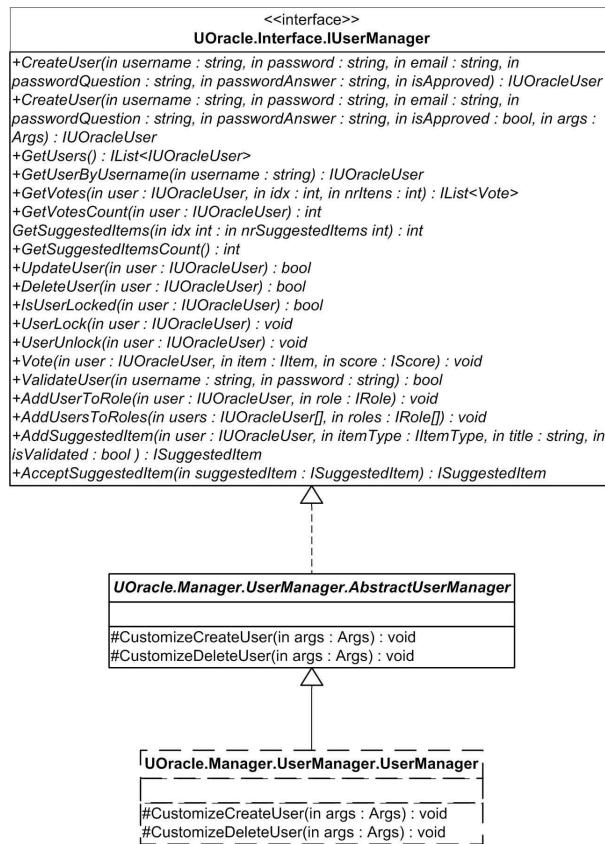


Figura 3.3: Diagrama de classes do gestor de utilizadores (*UserManager*)

**Personalização do gestor de utilizadores.** Para que o sistema possa ser utilizado por clientes com modelos de negócio distintos, com especificidades que não podem ser previstas na concepção do sistema, é contemplado um mecanismo que possibilita personalizar as entidades existentes. No exemplo que se segue, é descrito o princípio de desenho utilizado na personalização do gestor de utilizadores, sendo o mesmo princípio aplicado aos gestores de papéis, de itens e de recomendações. São revelados alguns pormenores de implementação, necessários à plena compreensão dos aspectos funcionais.

A personalização do gestor de utilizadores, deve ser efectuada através da extensão da classe *AbstractUserManager*, com a implementação dos métodos abstractos *CustomizeCreateUser* e *CustomizeDeleteUser*. Dessa forma são mantidas todas as funcionalidades disponibilizadas pelo sistema, podendo o cliente adicionar características próprias do seu modelo de negócio. Seguindo o padrão *Template Method* [8], os métodos *ganco*<sup>1</sup> *CustomizeCreateUser* e *CustomizeDeleteUser* são invocados pelo sistema nos processos de criação e eliminação do utilizador, dando a possibilidade ao cliente de acrescentar a sua lógica de negócio. Para que o cliente possa transportar os dados respeitantes à personalização para o sistema, é criada a interface *UOracle.Interface.IArgs*. Esta interface, presente na figura 3.4, define a propriedade *Identifier*, representando o identificador único (e.g., a chave primária) da entidade a personalizar, e a propriedade *Connection* que possibilita partilhar uma ligação à base de dados. O transporte é efectuado através do parâmetro de entrada do método *ganco*, possibilitando o envio de informação adicional a ser tratada na respectiva implementação.



Figura 3.4: Diagrama de classes da interface *IArgs*

Para clarificar o processo de personalização, é apresentado o fluxo de execução do processo de criação de um novo utilizador:

1. O Cliente cria a classe *MyUserPersonalData*, uma implementação de *IArgs*, definindo os campos que pretende acrescentar ao sistema;
2. Estende a classe *AbstractUserManager* e implementa o método abstracto *CustomizeCreateUser* (*hook method*);
3. Invoca o método *CreateUser* de *AbstractUserManager* (*template method*), enviando como parâmetro adicional a instância de *MyUserPersonalData*;

<sup>1</sup>Hook method, tal como definido no padrão *Template Method* [8].

4. O método *CreateUser* de *AbstractUserManager* inicia uma transacção, cria o utilizador no sistema e afecta as propriedades *Identifier* e *Connection* da instância de *MyUserPersonalData*. Automaticamente invoca o método gancho *CustomizeCreateUser* e completa a transacção.
- (a) A correr na mesma transacção, a implementação de *CustomizeCreateUser* efectuada pelo cliente, recebe a instância de *MyUserPersonalData* e, utilizando a camada de acesso a dados do cliente, cria o registo com a informação complementar. Realça-se o facto de ser reutilizada a ligação obtida através da propriedade *Connection*. O identificador único é utilizado para estabelecer uma ligação entre o registo e a entidade que se pretende personalizar (e.g., através de uma chave estrangeira).

A figura 3.5 ilustra uma hipotética hierarquia de classes desenhada pelo cliente, designada *ClientLibrary*, que exemplifica a extensão do sistema.

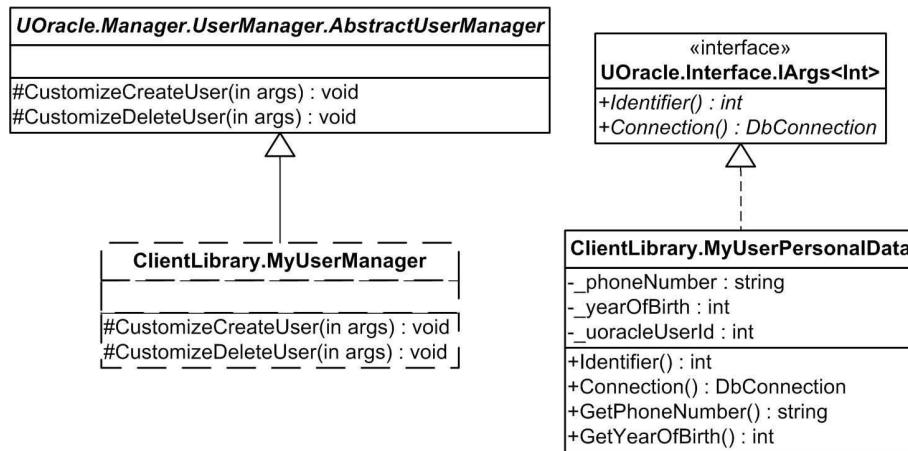


Figura 3.5: Diagrama de classes da biblioteca do cliente (*ClientLibrary*)

As listagens 3.1 e 3.2, mostram a implementação da biblioteca *ClientLibrary*. O cliente pretende acrescentar à sua lógica de negócio, como dados complementares, o ano de nascimento e o contacto telefónico dos utilizadores. Assim, começa por definir a classe *MyUserPersonalData* (implementando a interface *IArgs*), contendo a estrutura de dados que permitirá armazenar em memória os respectivos campos. Em seguida, define a classe *MyUserManager* (derivada de *AbstractUserManager*), e implementa os métodos gancho *CustomizeCreateUser* e *CustomizeDeleteUser* onde utiliza a sua camada de acesso a dados (*MyDal*) para: tornar persistentes os dados complementares contidos na implementação de *IArgs*; eliminar do repositório de persistência os dados complementares do utilizador. A invocação dos métodos *CreateUser* e *DeleteUser*, passando como argumento a instância de *MyUserPersonalData* desencadeará o processo de personalização.

```

1 namespace ClientLibrary {
2     public class MyUserPersonalData : IArgs<int>{
3         private readonly string _phoneNumber;
4         private readonly int _yearOfBirth, _uoracleUserId;
5         public MyUserPersonalData(string phoneNumber, int yearOfBirth){
6             _phoneNumber = phoneNumber;
7             _yearOfBirth = yearOfBirth;
8         }
9         public string GetPhoneNumber(){return _phoneNumber;}
10        public int GetYearOfBirth(){return _yearOfBirth;}
11        public int Identifier{
12            get{return _uoracleUserId;}
13            set{uoracleUserId = value;}
14        }
15    }
16 }
```

Listagem 3.1: Exemplo de implementação de *IArgs*

```

1 namespace ClientLibrary {
2     public sealed class MyUserManager : AbstractUserManager{
3         protected override void CustomizeCreateUser(IArgs<int> args){
4             MyUserPersonalData myPersonalData = (MyUserPersonalData)args;
5             using(MyDal myDal = new MyDal()){
6                 UserPersonalData userPersonalData = new UserPersonalData();
7                 userPersonalData.phoneNumber = myPersonalData.GetPhoneNumber();
8                 userPersonalData.yearOfBirth = myPersonalData.GetYearOfBirth();
9                 userPersonalData.uoracleUserId = args.Identifier;
10                myDal.AddToUserPersonalData(userPersonalData);
11                myDal.SaveChanges();
12            }
13        }
14        protected override void CustomizeDeleteUser(IArgs<int> args){
15            MyUserPersonalData myPersonalData = (MyUserPersonalData)args;
16            using(MyDal myDal = new MyDal()){
17                UserPersonalData userPersonalData = myDal.UserPersonalData.
18                    FirstOrDefault(p=>p.uoracleUserId==myPersonalData.Identifier);
19                myDal.DeleteObject(userPersonalData);
20                myDal.SaveChanges();
21            }
22        }
23    }
```

Listagem 3.2: Exemplo de personalização de *UserManager*

### Gestor de papéis (*RoleManager*)

Responsável pelo registo de papéis e respectiva associação a utilizadores. Esta hierarquia de classes complementa a segurança do sistema providenciando autorização baseada em papéis e disponibiliza as operações *CRUD*<sup>2</sup> necessárias à sua gestão.

A hierarquia de classes *RoleManager*, representada na figura 3.6, segue o mesmo princípio de desenho de *UserManager*, podendo o cliente delegar a responsabilidade de gestão de papéis no sistema através da classe *RoleManager*, fazer uma personalização através da classe *AbstractRoleManager*, ou utilizar *IRoleManager* caso pretenda uma implementação de raiz.

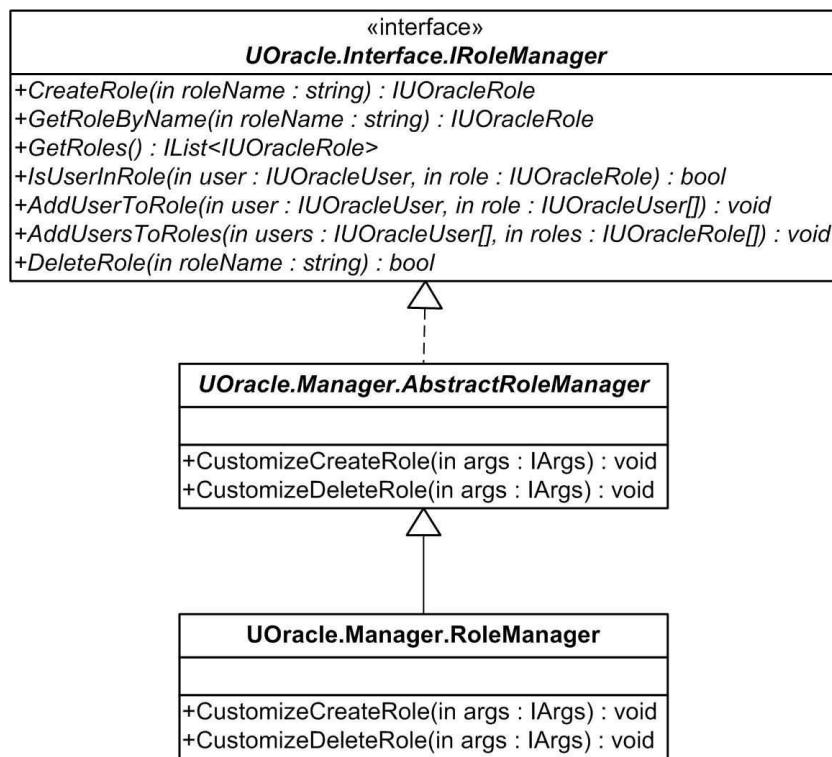


Figura 3.6: Diagrama de classes do gestor de papéis (*RoleManager*)

**Personalização do gestor de papéis.** A personalização deve ser efectuada através da extensão da classe *AbstractRoleManager* com a implementação dos métodos abstractos *CustomizeCreateRole* e *CustomizeDeleteRole*.

### Gestor de itens (*ItemManager*)

Responsável pelo registo de itens e respectiva catalogação, através da definição de tipos e categorias a associar.

<sup>2</sup>Create, Retrieve, Update e Delete.

A hierarquia de classes *ItemManager*, representada na figura 3.7, segue o mesmo princípio de desenho de *UserManager*, podendo o cliente delegar a responsabilidade de gestão de itens no sistema através da classe *ItemManager*, fazer uma personalização através da classe *AbstractItemManager*, ou utilizar *ItemManager* caso pretenda uma implementação de raiz.

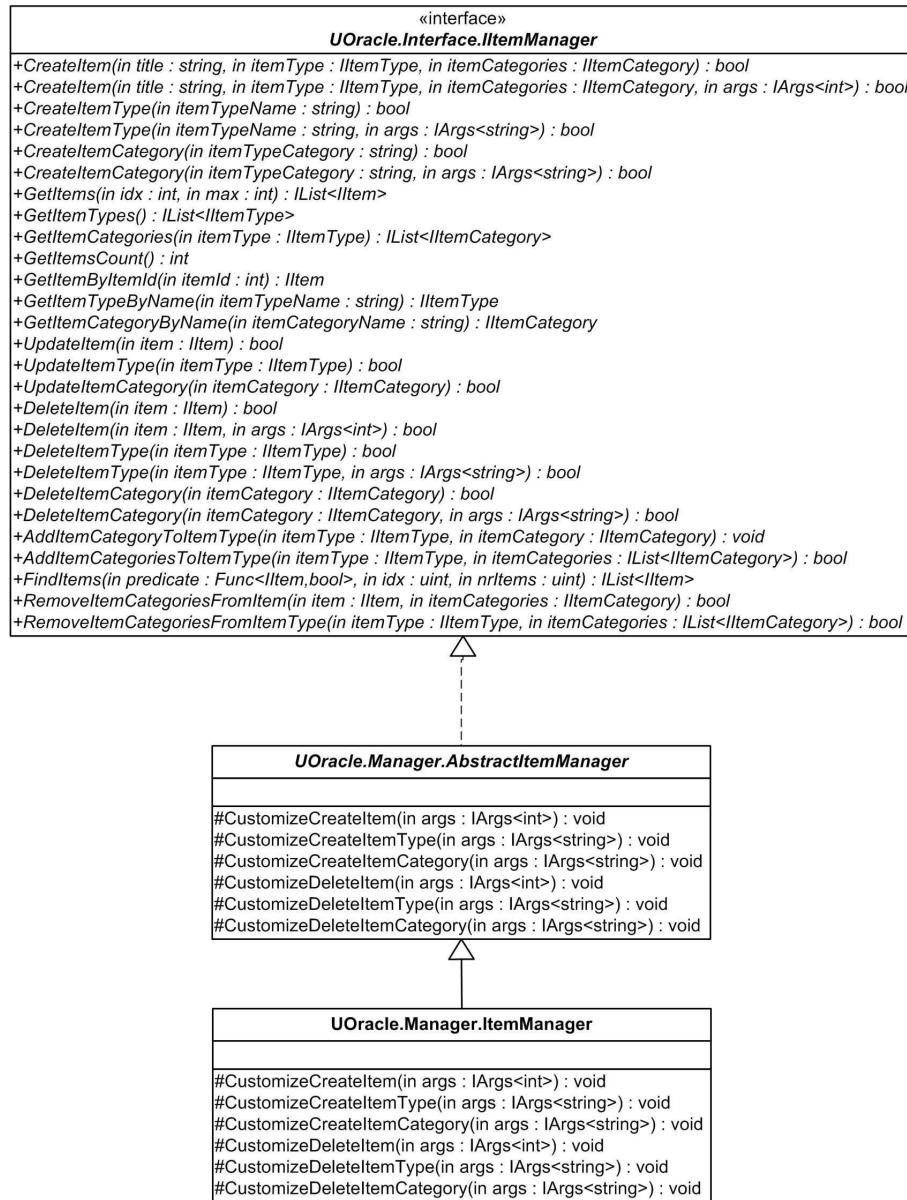


Figura 3.7: Diagrama de classes do gestor de itens (*ItemManager*)

**Personalização do gestor de itens.** A personalização deve ser efectuada através da extensão da classe *AbstractItemManager* com a implementação dos métodos abstractos *CustomizeCreateItem* e *CustomizeDeleteItem*.

Para personalização de um item, o cliente pode, por exemplo, acrescentar o campo *barCode* (associando um código de barras ao item), e redefinir os métodos *CustomizeCreateItem* e *CustomizeDeleteItem*. Seguindo o mesmo critério utilizado na personalização do utilizador, acrescenta a sua lógica de negócio.

A personalização de tipos e categorias segue o mesmo princípio, sendo efectuada através da implementação dos métodos *CustomizeCreateItemType*, *CustomizeDeleteItemType*, *CustomizeCreateItemCategory* e *CustomizeDeleteItemCategory*.

Com o objectivo de flexibilizar as operações de pesquisa de itens, é incluído o método *IList<IItem> FindItems(Func<IItem, bool> predicate, uint idx, uint nrItens)* que possibilita ao cliente efectuar uma pesquisa personalizada de itens segundo um critério definido por si. Este método devolve o conjunto de itens sobre os quais o predicado se verifica.

### Gestor de recomendações (*RecommendationManager*)

Responde às solicitações de *recomendação* dos utilizadores. Na hierarquia de classes *RecommendationManager* representada na figura 3.8 consta a interface *IRecommendationManager*.

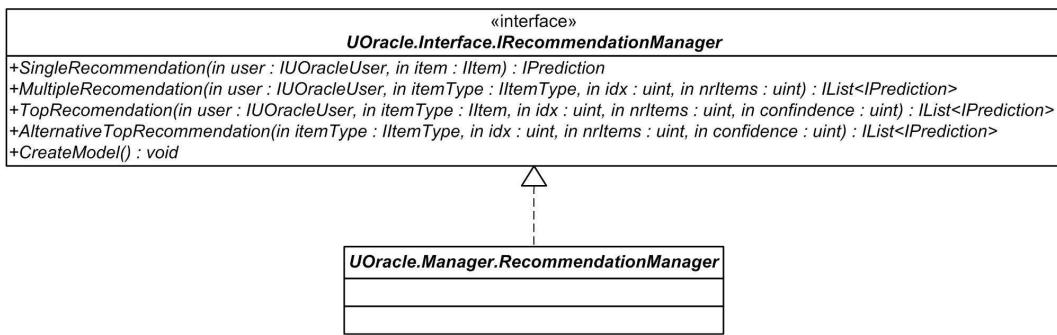


Figura 3.8: Diagrama de classes do gestor de recomendações (*RecommendationManager*)

Segue-se a definição do conjunto de métodos disponibilizados pelo gestor de recomendações:

**SingleRecommendation** - Dado um utilizador e um item, devolve uma predição;

**MultipleRecommendation** - Dado um utilizador e um tipo, devolve uma lista de predições ordenada de forma descendente pelo identificador do item. Esta ordenação privilegia os itens mais recentes. Os restantes argumentos do método são utilizados no processo de paginação para carregamento parcial de registo;

**TopRecommendation** - Dado um utilizador e um tipo, devolve uma lista de *predições* ordenada pelo grau de predição do item e, como segundo critério de ordenação, pelo número de votações contempladas na geração do modelo a que se atribui a designação de grau de confiança (**confidence**). Apenas as predições com um grau de confiança igual ao superior ao estipulado no parâmetro de entrada *confidence* são incluídas na resposta. Os restantes argumentos do método são utilizados no processo de paginação para carregamento parcial de registo;

**AlternativeTopRecommendation** - Devolve uma lista de predições contendo os itens mais votados pelos utilizadores do sistema, sendo o tipo desses itens especificado no parâmetro de entrada *itemType*. Os restantes argumentos do método são utilizados no processo de paginação para carregamento parcial de registo;

**CreateModel** - Inicia o processo de criação do modelo.

O cliente pode escolher entre as seguintes opções: utilizar a implementação de *IRecommendationManager* disponibilizada pelo sistema através da classe *RecommendationManager*, delegando nesta classe a interacção com o algoritmo de recomendação; realizar a sua própria implementação de *IRecommendationManager*, ficando à sua responsabilidade a interacção com o algoritmo de recomendação.

### Gestor de configurações (*ConfigurationManager*)

Para garantir a flexibilidade do sistema, é colocado à disposição do cliente a classe *ConfigurationManager*, a qual lhe dá autonomia na configuração do sistema para que este se ajuste ao seu modelo de negócio. O diagrama de classes de *ConfigurationManager* está ilustrado na figura 3.9.

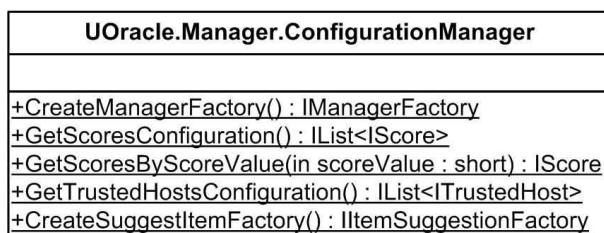


Figura 3.9: Diagrama de classes do gestor de configurações (*ConfigurationManager*)

Através do método estático *GetScoresConfiguration*, o cliente obtém uma lista de pontuações, passíveis de serem atribuídas a itens no processo de votação. O cliente deve definir essas pontuações através do ficheiro de configuração da aplicação.

Invocando o método estático *CreateManagerFactory*, o cliente obtém uma implementação de *IManagerFactory*. Esta interface define métodos para obter instâncias de gestores de utilizadores, papéis, itens, e recomendações. Caso o cliente não pretenda utilizar os gestores definidos por omissão no sistema, pode fornecer a sua própria fábrica de gestores mediante a implementação da interface *IManagerFactory*, obtida em tempo de execução através de reflexão. A figura 3.10 ilustra este processo.

Os restantes métodos estáticos estão directamente relacionados com o mecanismo de sugestão de itens e serão justificados no ponto 3.3.2.

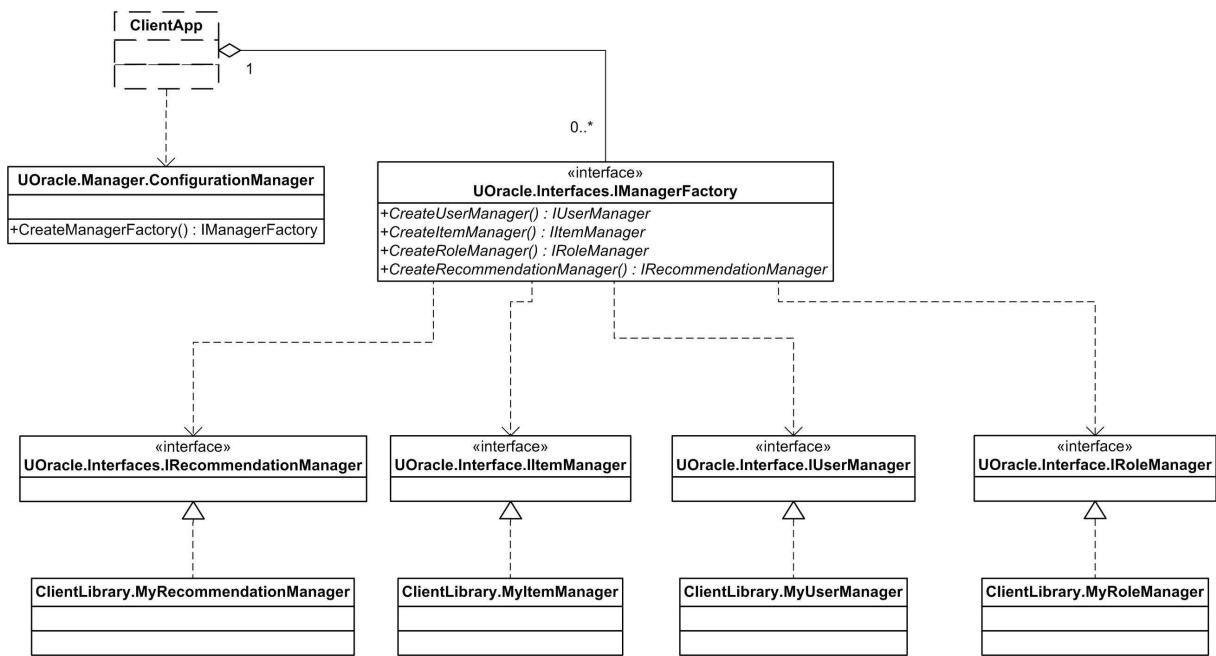


Figura 3.10: Diagrama de classes da fábrica de gestores (*ManagerFactory*)

O excerto do ficheiro de configuração *app.config* presente na listagem 3.3 exemplifica uma possível configuração do sistema. Na linha 3 é definida a secção *Scores*, e nas linhas 6-11 o respectivo conjunto de pontuações. Nas linhas 14 e 15 são adicionadas as chaves necessárias para identificar o *assembly* que define, neste caso, a fábrica de gestores *MyManagerFactory* implementada pelo cliente.

```

1 <configuration>
2   <configSections>
3     <section name="Scores" type="UOracle.Manager.ConfigurationManager.
4       ScoresConfigurationHandler,UOracle.Manager"/>
5   </configSections>
6   <Scores>
7     <Score name="Não Votado" value="0"/>
8     <Score name="Péssimo" value="1"/>
9     <Score name="Mau" value="2"/>
10    <Score name="Razoável" value="3"/>
11    <Score name="Bom" value="4"/>
12    <Score name="Excelente" value="5"/>
13  </Scores>
14  <appSettings>
15    <add key="ManagerAssembly" value="ClientLibrary"/>
16    <add key="ManagerFullName" value="ClientLibrary.MyManagerFactory"/>
17  </appSettings>
</configuration>
```

Listagem 3.3: Configuração de *Scores* e *ManagerFactory*

### Sugestão de itens (*ItemSuggestion*)

Para controlar o processo de sugestão de itens, é definida a interface *IItemSuggestion*. O único método desta interface, *MatchItem*, recebe como parâmetros o título e o tipo sugeridos para o item. A classe que implemente esta interface é responsável por devolver um booleano que indica se a sugestão foi verificada e considerada válida. Embora o sistema disponibilize uma implementação desta interface, foi idealizada a hierarquia de classes *ItemSuggestionFactory*, presente na figura 3.11, que possibilita ao cliente definir a sua própria fábrica de instâncias, sendo esta obtida em tempo de execução por reflexão mediante configuração no ficheiro *app.config*.

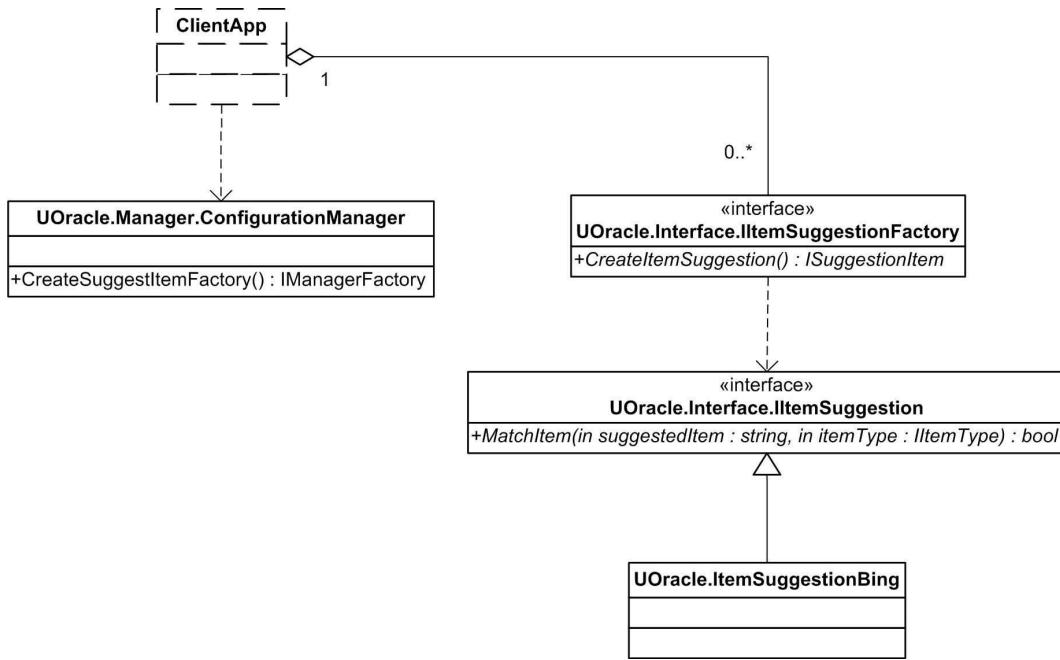


Figura 3.11: Diagrama de classes de *ItemSuggestionFactory*

**Trusted hosts.** O cliente pode definir, no ficheiro de configuração, um conjunto de *hosts* que, associados a um tipo de item, servem de referência no processo de sugestão. Esta informação pode ser utilizada pela implementação de *IItemSuggestion* para avaliar o grau de correcção relativamente ao item sugerido pelo utilizador. Segue-se um exemplo que ilustra uma possível utilização de um *trusted host*:

1. O cliente configurou o *trusted host* *imdb* como um *host* de referência para itens do tipo *Cinema*;
2. O utilizador sugere o título *Blade Runner*, associando-lhe o tipo *Cinema*;
3. A sugestão é processada pelo sistema que, utilizando um serviço de sugestão (e.g., *Bing* da *Microsoft*), analisa o resultado da pesquisa e compara-o com o *host* configurado;
4. Se o resultado da pesquisa referir o *host* *imdb* como sendo relevante, há uma probabilidade elevada de correcção na sugestão, e esta deve ser considerada.

Os *trusted hosts* podem ser obtidos em tempo de execução através do método estático *GetTrustedHostsConfiguration*, presente no gestor de configurações.

O excerto do ficheiro de configuração *app.config* presente na listagem 3.4 exemplifica uma possível configuração do sistema. Na linha 3 é definida a secção *TrustedHosts* e na linha 6 define-se um *host* de confiança. Nas linhas 9 e 10 são adicionadas as chaves necessárias para identificar o *assembly* que define a fábrica de *IItemSuggestion* disponibilizada pelo sistema.

```

1 <configuration>
2   <configSections>
3     <section name="TrustedHosts" type="UOracle.Manager.ConfigurationManager
4       .TrustedHostsConfigurationHandler,UOracle.Manager"/>
5   </configSections>
6   <TrustedHosts>
7     <TrustedHost hostName="www.imdb.com" itemType="Cinema"/>
8   </TrustedHosts>
9   <appSettings>
10    <add key="ItemSuggestionAssembly" value="UOracle.Manager"/>
11    <add key="ItemSuggestionFullName" value="UOracle.Manager.
12      ItemSuggestionFactory"/>
13  </appSettings>
14 </configuration>

```

Listagem 3.4: Configuração de *TrustedHosts* e *ItemSuggestionFactory*

## 3.4 Implementação

### 3.4.1 Introdução

Pretende-se documentar e justificar as decisões tomadas no que diz respeito à concretização do sistema, bem como às tecnologias utilizadas para dar suporte a essa concretização. Sendo este um projecto académico com um período curto de implementação, foram aplicados, sempre que pertinente, conceitos adquiridos durante o período de licenciatura, na mesma tecnologia em que foi abordada a sua componente prática. Dessa forma, rentabilizou-se o tempo disponível, consolidando-se os conhecimentos adquiridos.

Optou-se pela utilização da *Framework .NET* na sua versão 3.5 (linguagem *C#*) na concretização das bibliotecas de código. Esta escolha deveu-se essencialmente aos seguintes factores:

- Gestão automática de memória e robustez de código;
- Isolamento e portabilidade garantido pelo ambiente virtual de execução;
- Integração do processo de autenticação e autorização baseado em *providers* (*MembershipProvider* e *RoleProvider*);
- Facilidade na criação e distribuição de componentes *web* (*Custom Controls*);
- Suporte para a construção de aplicações *web*, contributo importante para o desenvolvimento do protótipo da aplicação.

### 3.4.2 Organização do código fonte

A figura 3.12, apresenta a organização de código utilizada na implementação do sistema de recomendações. O espaço de nomes *UOracle*, raiz deste projecto, contém os subespaços: *Interface*, onde se encontram as interfaces públicas do sistema; *Manager*, onde se encontram as implementações dos diferentes gestores e a implementação da camada de acesso a dados.

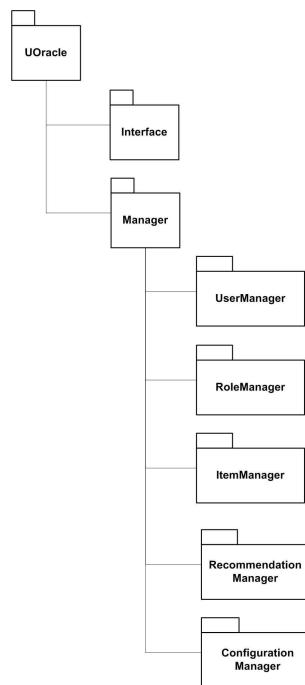


Figura 3.12: Representação dos espaço de nomes do sistema

### 3.4.3 Camada de acesso a dados (*DAL - Data Access Layer*)

Para armazenar informação de forma persistente, é conveniente tornar transparente o acesso à fonte de dados. Assim, minimiza-se a dependência, simplificando-se eventuais reestruturações na forma como os dados são acedidos. A *DAL* cria uma abstracção entre o modelo físico e o modelo lógico, ou seja, é possível alternar entre diversas fontes de dados sem a necessidade de reestruturar a lógica de negócio, independentemente da estrutura do modelo físico. A figura 3.13 ilustra os níveis de abstracção no acesso a dados.

A utilização de uma camada de acesso a dados é considerada uma boa prática, mas a sua implementação é um processo complexo. A *ADO.NET Entity Framework* da *Microsoft* e a *NHibernate* ([www.hibernate.org](http://www.hibernate.org)) são duas tecnologias que permitem construir camadas de acesso a dados, cuja arquitectura providencia abstracção relativamente ao modelo físico.

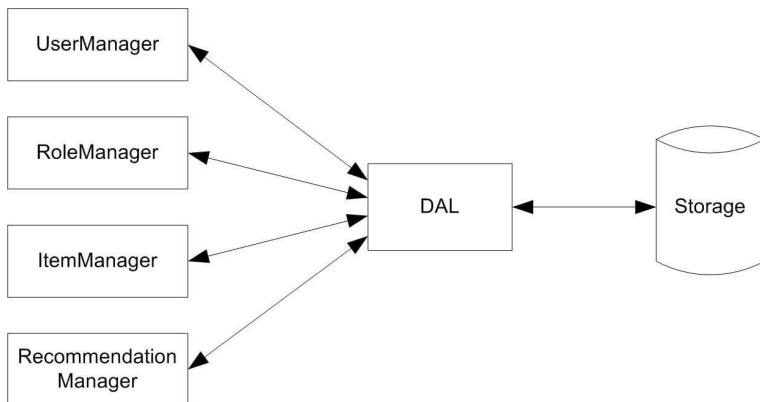


Figura 3.13: Camada de acesso a dados

Por não existir um contacto prévio com as tecnologias referidas, e devido à escassez de tempo, necessário para estabelecer uma comparação efectiva entre ambas, foi utilizada a solução da *Microsoft* devido à garantia de suporte e integração. Considerando as características semelhantes das duas soluções, importa salientar que a *framework NHibernate* é um projecto *open source* suportado por uma vasta comunidade de programadores, factor que pesaria no processo de decisão de um projecto com objectivos comerciais.

### **ADO.NET Entity Framework**

Um dos objectivos primordiais da mais recente versão do *ADO.NET* é aumentar o nível de abstracção no acesso a dados persistentes, oferecendo uma alternativa à tradicional utilização de *DataReaders* e *DataSets*. A maioria das aplicações desenvolvidas têm um modelo conceptual de dados que representa as entidades envolvidas e a forma como estas interagem entre si. Uma vez que é frequente a utilização de bases de dados relacionais, existe uma dificuldade acrescida na sua tradução para o modelo orientado a objectos utilizado ao nível da aplicação, e vice-versa.

O modelo de dados gerado automaticamente pela *Entity Framework*, designado *EDM (Entity Data Model)*, pode ser consultado no apêndice A.

**Extensão das entidades de negócio.** As classes geradas automaticamente pela *Entity Framework*, também designadas por entidades de negócio, são classes parciais. Esta particularidade possibilita o complemento das entidades através da criação de uma classe parcial com a mesma designação, contendo a restante definição da classe. Foi explorada esta característica na redefinição dos métodos: *ToString*, para facilitar a apresentação nas interfaces gráficas; *Equals* e *GetHashCode* (por convenção) por forma a personalizar a comparação entre instâncias do mesmo tipo e evitar colisões em tabelas de *hash*.

Para providenciar abstracção sobre os tipos concretos, é adicionada uma interface por cada entidade de negócio, disponibilizando apenas os seus métodos públicos. Na figura 3.14 apresenta-se como exemplo a hierarquia da entidade *UOracleUser* que representa o utilizador do sistema.

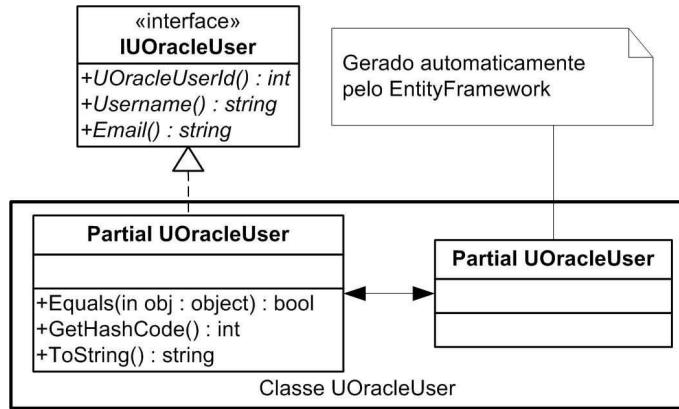


Figura 3.14: Diagrama de classes da hierarquia *UOracleUser*

Segue-se a enumeração dos passos necessários na criação da entidade *UOracleUser*:

1. Criação da interface *IUOracleUser* expondo as propriedades *UOracleUserId*, *Username*, e *Email*;
2. Criação da classe parcial *UOracleUser* implementando *IUOracleUser*;
3. Redefinição em *UOracleUser* dos métodos *ToString*, *Equals*, e *GetHashCode*.

**Restrição na associação de categorias a itens.** Como já referido em 3.3.1, as restrições impostas pelo modelo relacional não são suficientes para garantir a associação de categorias a itens de uma forma coerente. Para restringir programaticamente actualizações erróneas foi utilizado um gatilho *Instead Of* nas operações de *INSERT* e *UPDATE*. Realça-se o facto de, implicitamente, não existir recursividade directa nos gatilhos do tipo *Instead Of*. Na listagem 3.5 consta o código *T-SQL* correspondente à implementação do gatilho *TriggerCategories*.

```

1 create TRIGGER TriggerCategories ON Item_ItemTypeItemCategory
2 --triggers instead of não provocam chamada recursiva
3 INSTEAD OF INSERT, UPDATE
4 AS
5 if (update(itemType) or update(itemCategory) or update(itemId))
6 begin
7 insert into Item_ItemTypeItemCategory(itemType, itemCategory, itemId)
8 select item.itemType, i.itemCategory, item.itemId from item inner join
9     inserted as i
9 on item.itemType=i.itemType and item.itemId = i.itemId
10 end

```

Listagem 3.5: Implementação do gatilho *TriggerCategories*

Como alternativa à utilização do gatilho, a restrição poderia ter sido imposta através da criação de procedimentos armazenados, posteriormente importados para o modelo gerado pela *Entity Framework* e associados às operações *CRUD* da entidade item. Contudo seria necessário restringir o acesso directo às tabelas através de permissões, já que uma alteração efectuada directamente quebraria a lógica de negócios.

Restringir o acesso directo a tabelas, para que este possa ser efectuado por meio de vistas e procedimentos armazenados mediante configuração de permissões, é considerado uma boa prática. No âmbito deste projecto não foram aplicadas tais medidas de segurança, que seriam indispensáveis num cenário real.

### 3.4.4 *UserManager* e *RoleManager*

A autenticação e autorização de utilizadores é um requisito da maioria das aplicações, particularmente nas aplicações *web* em que os conteúdos são disponibilizados consoante as credenciais do utilizador. Sendo este projecto implementado em C#, assumindo-se um compromisso com a *Framework .NET*, procurou-se explorar algumas das suas mais valias, de que é exemplo o sistema integrado de autenticação e autorização baseado em *providers*.

#### *Membership Provider* e *Role Provider*

A *Framework .NET* disponibiliza um sistema integrado de autenticação e autorização (*role-based*) baseado em *providers*, respectivamente *MembershipProvider* e *RoleProvider*. Estas duas classes são abstractas, existindo implementações que possibilitam a persistência de dados no *SGBD SQL Server*, mais concretamente *SqlMembershipProvider* e *SqlRoleProvider*. As tabelas são geradas automaticamente, bem como um conjunto de procedimentos armazenados utilizados para realizar as operações *CRUD* necessárias à gestão de utilizadores e de papéis.

Para facilitar a utilização dos *providers*, a Framework .NET disponibiliza as classes *Membership* e *Roles*, que implementam o padrão *Façade* [8]. O método *CreateUser* da classe *Membership*, ilustrado na listagem 3.7, simplifica a utilização do método *CreateUser* da classe abstracta *MembershipProvider* ilustrada na listagem 3.6. A classe *Membership* encarrega-se de criar as instâncias necessárias para a invocação do método *CreateUser* de *MembershipProvider* com os parâmetros necessários.

```
1 public abstract MembershipUser CreateUser(
2     string username ,
3     string password ,
4     string email ,
5     string passwordQuestion ,
6     string passwordAnswer ,
7     bool isApproved ,
8     Object providerUserKey ,
9     out MembershipCreateStatus status
10 )
```

Listagem 3.6: Método *CreateUser* de *MembershipProvider*

```
1 public static MembershipUser CreateUser(
2     string username ,
3     string password
4 )
```

Listagem 3.7: Método *CreateUser* da classe *Membership*

**Application name.** O modelo relacional destes dois *providers* possibilita armazenar informação única por aplicação. Assim, a título de exemplo, podem coexistir dois utilizadores com o mesmo *username* desde que sejam de aplicações distintas. Como alternativa, múltiplas aplicações podem partilhar os mesmos utilizadores desde que especifiquem o mesmo nome de aplicação no ficheiro de configuração.

Neste projecto, definiu-se que diferentes aplicações partilham os mesmos registo, não sendo contemplado no modelo relacional (das entidades *UOracle*) um atributo que identifique inequivocamente a aplicação.

**Independência e extensão de capacidades.** Para garantir a independência relativamente às tabelas disponibilizadas pelo *SqlMembershipProvider*, foram criadas tabelas congêneres, de que é exemplo *UOracleUser*. Esta tabela contém uma chave estrangeira para *aspnet\_Membership*. Desta forma, tabelas que necessitam de referenciar utilizadores podem fazê-lo através de *UOracleUser*, minimizando-se a dependência relativamente ao *SqlMembershipProvider*.

No contexto deste projecto, houve necessidade de armazenar informação adicional sobre os utilizadores. Embora a *Framework .NET* disponibilize um *provider* para esse efeito, designado *Profile*, que possibilita definir um conjunto de propriedades associadas aos utilizadores, bem como tipos complexos (e.g., Contactos), que encapsulam propriedades (e.g., Morada, Telefone), decidiu-se neste caso por se utilizar uma estratégia diferente, que pode ser aplicada genericamente a qualquer outra entidade que se pretenda personalizar. Como exemplo, é explicada a extensão da entidade que representa o utilizador:

Foi criada a tabela *UOracleUser* para armazenamento de informação adicional sobre o utilizador, contendo como chave estrangeira o atributo *UserId* da tabela *aspnet\_Membership*, e um identificador único do tipo inteiro (de incremento automático). Com esta medida, minimizou-se a dependência relativamente ao modelo de dados do *SqlMembershipProvider*.

Os métodos *CreateUser* e *DeleteUser* de *AbstractUserManager* são responsáveis por controlar o processo de inserção e remoção do utilizador. Na listagem 3.8 é apresentado um excerto do código fonte de *CreateUser*, onde após a criação do utilizador na tabela *aspnet\_Membership*, na linha 5, é criado o registo correspondente na tabela *UOracleUser*, na linha 10. A atomicidade desta operação é garantida através da transacção iniciada na linha 2. Importa salientar que por não ser possível partilhar a conexão entre os dois *providers* (*MembershipProvider* e *Entity Framework*), no processo de actualização de dados, visível na linha 12, a transacção é promovida a transacção distribuída apesar de ser efectuada na mesma máquina.

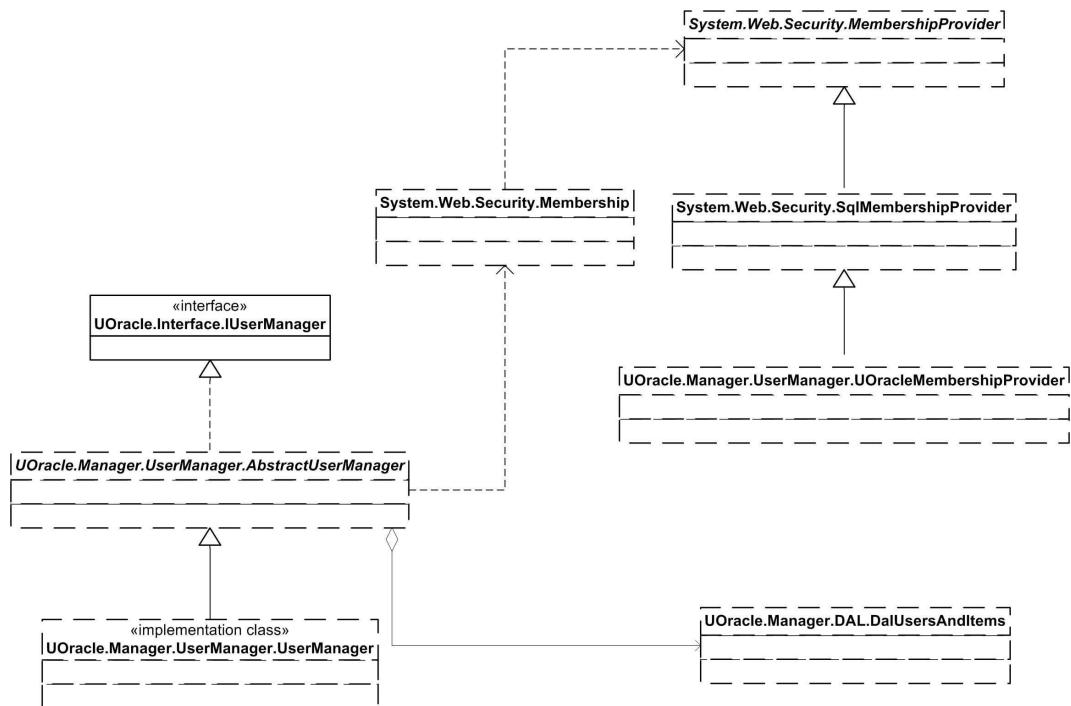
Foi criada a classe *UOracleMembershipProvider*, estendendo de *SqlMembershipProvider*. Esta medida é puramente conceptual, uma vez que não é redefinido qualquer método. No entanto, privilegia futuras implementações, minimizando a dependência do sistema em relação ao *SqlMembershipProvider*. A mesma estratégia foi aplicada na criação da classe *UOracleRoleProvider*, responsável pela associação de utilizadores a papéis.

Na figura 3.15 está representada a hierarquia de classes *UserManager*, bem como a integração do sistema de autenticação facultada pela classe *SqlMembershipProvider*.

Na figura 3.16 está representada a hierarquia de classes *RoleManager*, bem como a integração do sistema de autorização facultada pela classe *SqlRoleProvider*.

```

1 // inicia transacção
2 using (TransactionScope transactionScope = new TransactionScope()){
3 ...
4 // cria utilizador na tabela aspnet_Membership
5 MembershipUser membershipUser = Membership.CreateUser(username, password
6 , email, passwordQuestion, passwordAnswer, isApproved, out status);
7 ...
8 ...
9 // cria utilizador na tabela UOracleUser
10 dal.AddToUOracleUser(user);
11 // inicia uma transacção distribuída
12 dal.SaveChanges();
13 ...
14 // completa a transacção
15 transactionScope.Complete();
16 ...
17 }
18 }
```

Listagem 3.8: Método *CreateUser* da classe *AbstractUserManager*Figura 3.15: Diagrama de classes da implementação de *UserManager*

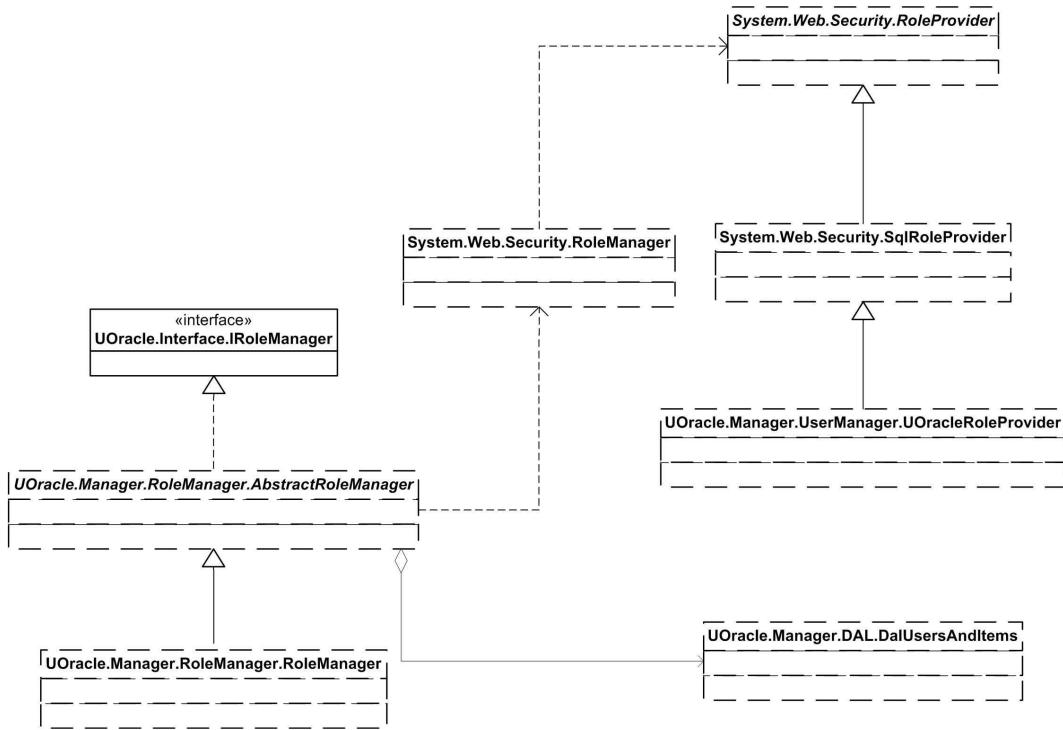


Figura 3.16: Diagrama de classes da implementação de *RoleManager*

### 3.4.5 *ItemManager*

A implementação desta hierarquia de classes ficou confinada às operações *CRUD* que suportam a gestão de itens, e à implementação do método *FindItems* concebido para realizar pesquisas sobre itens. Este último recebe o *delegate Func* (o predicado), parametrizado com *IItem* e um booleano. Segue-se o fluxo de execução que clarifica o processo de pesquisa:

1. O cliente define um método de pesquisa que obedece à assinatura do *delegate Func*;
2. Quando o método é invocado através do *delegate*, avalia o item segundo o predicado definido;
3. São retornados os itens que satisfazem o predicado.

Os parâmetros *idx* e *nrItens* são utilizados pelo processo de paginação para efectuar o carregamento parcial dos itens que satisfazem o predicado.

### 3.4.6 *RecommendationManager*

O gestor de recomendações controla o processo de recomendação. É responsável por invocar os procedimentos armazenados que: iniciam a geração do modelo de recomendações; processam os pedidos de recomendação solicitadas pelos utilizadores. Todo o processamento é efectuado

pelo *SGBD*, minimizando o tráfego de rede. O facto de o algoritmo ser executado no mesmo processo do *SGBD*, operando directamente sobre a fonte de dados, eleva o grau de desempenho do sistema. Apenas o resultado desse processamento é transportado para a aplicação, ou seja, a resposta ao pedido de recomendação do utilizador. O diagrama de sequência representado na figura 3.17 ilustra o processo de recomendação.

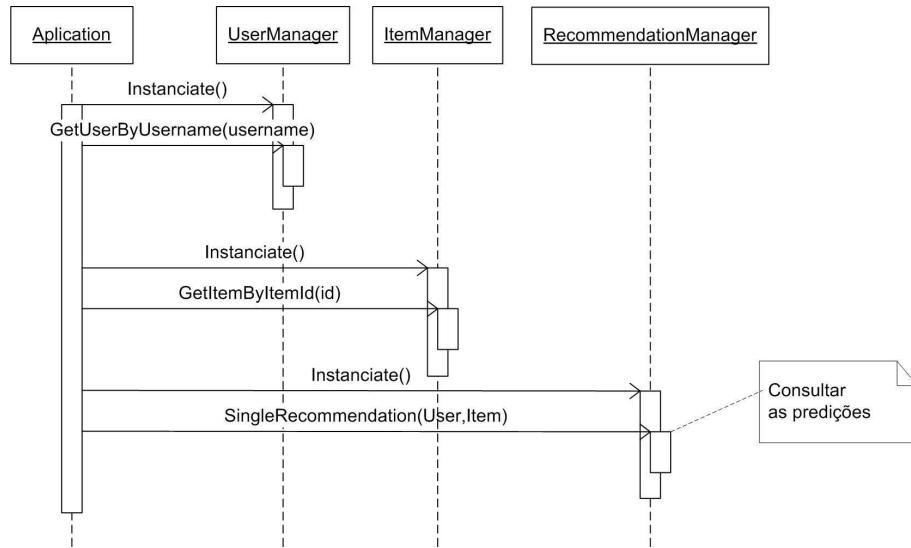


Figura 3.17: Diagrama de sequência do processo de recomendação

Segue-se o fluxo de execução deste processo:

1. A aplicação utiliza uma instância de `UserManager` (através da classe `ConfigurationManager`) para invocar o método  `GetUserByUsername`, obtendo a instância que representa o utilizador;
2. Utiliza uma instância de `ItemManager` para invocar o método `GetItemById`, obtendo a instância que representa o item seleccionado;
3. Obtém uma instância de `RecommendationManager` e invoca o método `SingleRecommendation` que retorna a previsão para o utilizador relativamente ao item.

### 3.4.7 Votação e sugestão de itens

A figura 3.18 apresenta o diagrama de sequência do processo de votação. Segue-se o fluxo de execução deste processo:

1. A aplicação usa uma instância de `ItemManager` (através da classe `ConfigurationManager`) para invocar o método `GetItemById`, obtendo a instância que representa o item a voto;

2. Através de uma instância de *UserManager*, invoca o método *GetUserId* para obter a instância que representa o votante;
3. Com estes dois elementos na sua posse invoca o método *Vote* de *UserManager*, enviando como parâmetro adicional a pontuação a atribuir ao item.

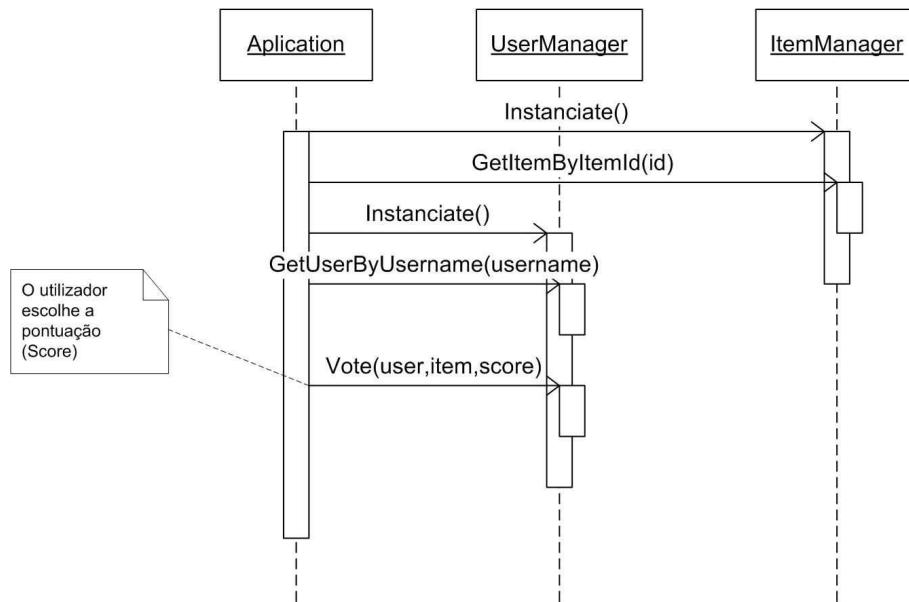


Figura 3.18: Diagrama de sequência do processo de votação

### Sistema automático de triagem (*ItemSuggestionBing*)

Para dar forma à triagem, que visa avaliar a sugestão de um item por parte do utilizador, recorreu-se aos serviços de pesquisa e sugestão *Bing* da *Microsoft*. Este serviço possibilita efectuar pesquisas e obter um conjunto de resultados que incluem sugestões de ortografia, sítios ordenados por ordem de relevância, etc.

A implementação da interface *IItemSuggestion*, disponibilizada pelo sistema para controlar o processo de sugestões de itens é suportada no *Bing Web Services* da *Microsoft*. A sugestão do utilizador é encaminhada para o mecanismo de validação que, recorrendo ao *web service*, efectua uma triagem automática. Em traços gerais, enumeramos o fluxo de execução deste processo:

1. O utilizador define o título e o tipo do item que pretende sugerir;
2. A sugestão é enviada para o mecanismo de validação;
3. É verificada a existência na base de dados de um item com o mesmo título e tipo. Caso se confirme a existência, o processo termina;

4. Caso não se confirme a existência do item na base de dados, o mecanismo de validação consulta o *web service Bing* e recolhe informação relativa aos campos de *spelling suggestion* (que identifica um possível erro de ortografia) e *hosts* (que disponibiliza *hosts*, por ordem de relevância, retornados pela pesquisa);
5. Caso o campo de *spelling suggestion* esteja vazio, mediante o tipo escolhido para o item, são carregados do ficheiro de configuração os endereços dos *hosts* de referência (e.g., *hostName="www.imdb.com" itemType="Cinema"*), sendo estes comparados com os *hosts* retornados pelo *web service*;
6. Se, e só se, o campo de *spelling suggestion* estiver vazio e os *trusted hosts* coincidirem com o resultado da pesquisa, a sugestão será marcada como validada antes de ser remetida para o sistema para posterior avaliação por parte do moderador.

Na figura 3.19 está patente o fluxo de execução do processo de sugestão de itens.

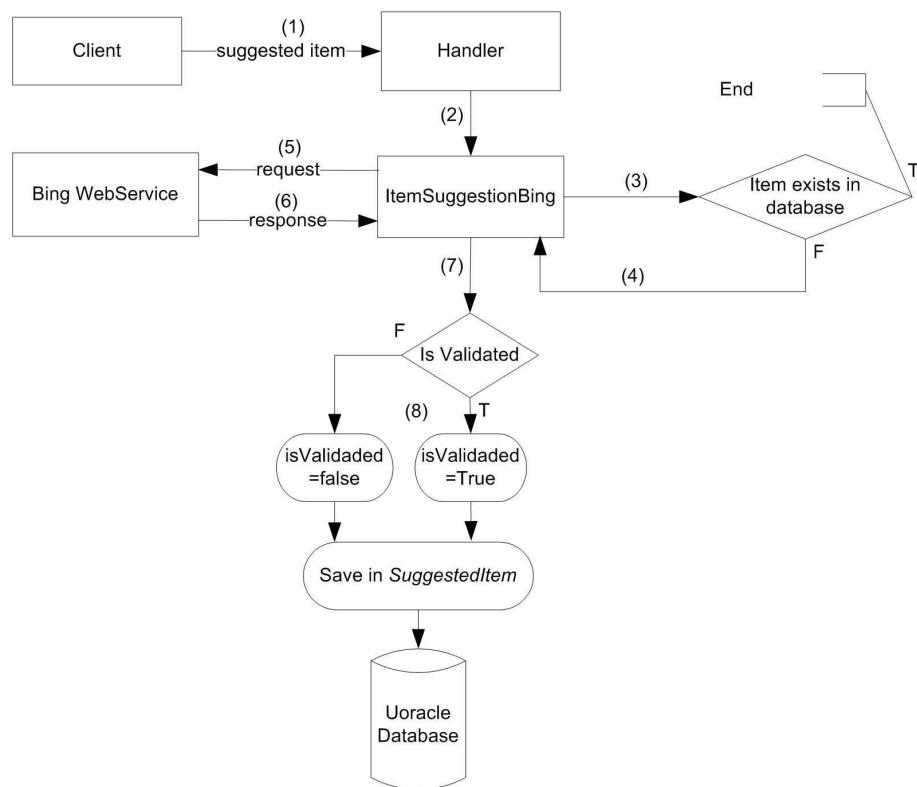


Figura 3.19: Fluxo de execução do processo de sugestão de itens

### 3.4.8 Replicação

Tal como referido em 3.4.8, o algoritmo que gera o modelo de recomendação deve ter acesso exclusivo aos perfis dos utilizadores. Uma forma de contornar este problema passa por utilizar

o mecanismo de replicação disponibilizado pelo *SQL Server*, não prejudicando o acesso das aplicações à tabela original. Dependendo da configuração do sistema, essa replicação pode ser efectuada na mesma máquina, ou entre máquinas diferentes caso seja utilizado o *SGBD* alternativo.

Foi criada, no *SQL Sever*, a publicação *VotePushReplication*, tendo como artigo a tabela *Vote* (que contém os perfis dos utilizadores). Sendo a replicação transaccional, as alterações de perfil fluem periodicamente para o assinante, neste caso, a base de dados que contém a réplica de *Vote*, utilizada pelo algoritmo que gera o modelo de recomendação.

### 3.4.9 Paginação

Nas operações que envolvem a leitura de uma quantidade elevada de dados para memória, optou-se pela disponibilização de métodos que possibilitam o carregamento parcial (de  $N$  em  $N$  registos), e que têm como parâmetros de entrada o índice de início e o número total de registos a ler. A optimização referida está presente nos métodos *GetVotes* de *UserManager* e *GetItems* de *ItemManager*. Este factor constitui uma mais valia para o processo de paginação. Os registos são filtrados pelo *SGBD* através de instruções *T-SQL*, sendo transferidos para a aplicação apenas os registos a apresentar por página. Contudo, os identificadores únicos nem sempre são sequenciais (e.g., um item pode ser apagado), o que dificulta o acesso aleatório ao primeiro registo de cada página. Segue-se um exemplo para clarificar a dificuldade enunciada e a estratégia adoptada na resolução do problema.

Considerando uma paginação que apresenta dez resultados por página, com um total de três páginas, pretende-se mostrar os registos correspondentes à segunda página. Se os identificadores únicos fossem sequenciais, o primeiro registo da segunda página teria como identificador único o valor *11*. Contudo, no caso dos dois primeiros registos terem como identificadores, respectivamente, os valores *1* e *3*, estando todos os outros ordenados sequencialmente, o primeiro registo da segunda página terá como identificador único o valor *12*. Enumeram-se os passos do algoritmo utilizado na resolução deste problema, adaptado ao presente exemplo:

1. Efectua-se uma projecção dos **10 + 1** primeiros itens, ordenados de forma ascendente pelo identificador único;
2. Sobre essa projecção extrai-se o identificador máximo para obter o primeiro registo da segunda página (identificador *12*);
3. Efectua-se uma segunda projecção, através da qual se obtém os *10* itens da segunda página, ordenados de forma ascendente pelo identificador único, ou seja, os *top(10)* a partir do registo obtido no ponto anterior.

A utilização deste método resolve a indexação aleatória, porém, apresenta um caso particular. Seguindo o exemplo anterior, se o registo da primeira página com o identificador *I* fosse apagado entre o carregamento das duas primeiras páginas, o registo com o identificador *I2* não seria visualizado na segunda página, uma vez que passaria a pertencer à primeira.

### 3.4.10 Testes unitários

Para garantir a fiabilidade dos métodos disponibilizados na biblioteca de classes, e a consistência necessária para que futuras alterações não comprometam o comportamento do sistema, é disponibilizado o conjunto de testes unitários utilizados na sua verificação juntamente com o código fonte.



# Capítulo 4

## Componentes *web*

### 4.1 Introdução

Com a massificação do acesso à Internet e as facilidades do acesso sem fios (e.g., *WiFi*, *3G*), as aplicações *web* passaram a ocupar um lugar de destaque. Considerando a conjuntura actual, define-se como prioridade disponibilizar mecanismos que facilitem o desenvolvimento deste tipo de aplicações, com o objectivo de minimizar o tempo despendido pelo cliente na sua implementação. Assim, é disponibilizado um conjunto de componentes .NET, mais especificamente *Custom Controls*, que podem ser utilizados na interacção com o utilizador. Sempre que pertinente, os componentes incorporam a técnica *AJAX (Asynchronous Javascript And XML)*. Desta forma, reduz-se o tráfego resultante dos pedidos e respostas (aplicação cliente/servidor), uma vez que apenas é efectuado o carregamento dos conteúdos que necessitam ser actualizados. O facto desta técnica evitar o *Postback*, que resultaria no carregamento integral da página, beneficia a interacção com o utilizador, transmitindo uma sensação de fluidez semelhante à interface de uma aplicação *desktop*.

São também disponibilizados *handlers*, isto é, classes implementando a interface *IHttpHandler*, que podem ser utilizados pelo cliente numa aplicação *web* para efectuar pedidos específicos ao servidor, por exemplo, pedir ao servidor as categorias correspondentes a um determinado tipo.

As implementações referidas estão disponíveis nos espaços de nomes:

*UOracle.WebComponents.CustomControls* e *UOracle.WebComponents.Handlers*.

Remete-se o leitor para o código fonte e respectiva documentação onde podem ser observados os detalhes de implementação.

## 4.2 *Handlers*

Para dar suporte aos pedidos ao servidor efectuados pelos controlos desenvolvidos, são implementados os *handlers* *Categories.ashx* e *Vote.ashx*.

**Categories.ashx** Este *handler* tem como parâmetro de entrada o tipo de um item e devolve uma *string*, em formato *JSON*, contendo todas as categorias associadas ao tipo especificado.

**Vote.ashx** Através da invocação deste *handler* é possível votar num determinado item. Apresenta como parâmetros de entrada o identificador único do item e a respectiva pontuação. Caso o identificador do item coincida com um dos itens existentes, e haja conformidade com as restrições de autenticação e autorização do sistema, é processado o voto do utilizador.

## 4.3 *Custom Controls*

### 4.3.1 *GridViewPagination*

A classe *System.Web.UI.WebControls.GridView* [13] disponível na *Framework .NET* possibilita a visualização e edição de registo provenientes de uma fonte de dados. Quando a quantidade de registo é elevada, é possível recorrer ao mecanismo de paginação do componente, isto é, apresentar  $N$  em  $N$  registo por página. Embora o *GridView* possibilite paginação, todos os registo são carregados em memória e não apenas os necessários para a visualização da página corrente. Face a este problema, documentado em [14], a Microsoft sugere como solução a utilização da classe *System.Web.UI.WebControls.ObjectDataSource* [15] que, através das suas propriedades, possibilita controlar o processo de paginação, carregando para memória apenas os itens necessários para a visualização da página corrente. Assim, foi criada a classe *GridViewPagination*, que deriva de *GridView*, acrescentando a paginação referida através do encapsulamento de um objecto do tipo *ObjectDataSource*. Salienta-se que, do objecto *ObjectDataSource*, apenas ficam expostas propriedades que possibilitam a configuração de paginação. Na figura 4.1 apresenta-se o diagrama de classes do referido componente.

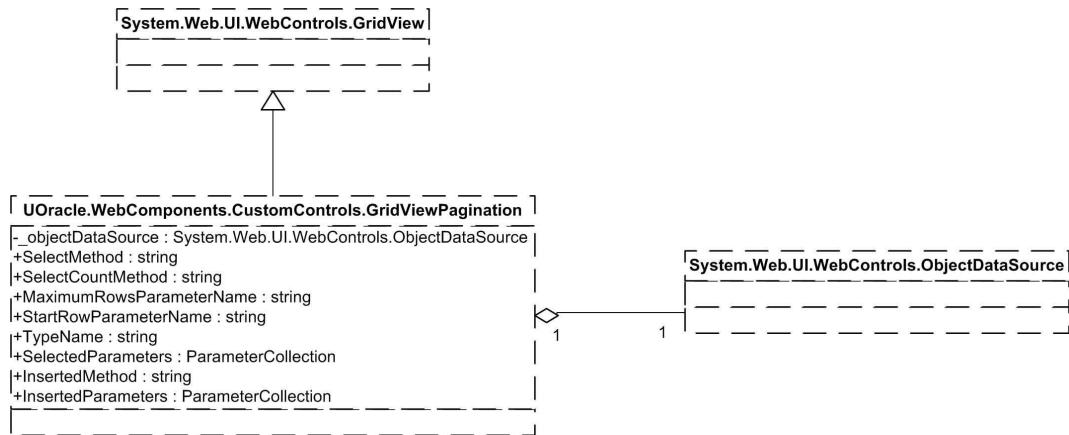


Figura 4.1: Diagrama de classes do componente *GridViewPagination*

Segue-se a descrição das propriedades que dão suporte à paginação:

**SelectMethod** - O nome do método que o *ObjectDataSource* invoca para obter os registos;

**SelectCountMethod** - O nome do método que o *ObjectDataSource* invoca para obter o número total de registos;

**MaximumRowsParameterName** - O nome do parâmetro que é usado para indicar o número máximo de registos a apresentar por página;

**StartRowParameterName** - O nome do parâmetro que é usado para indicar o valor do primeiro índice a apresentar na página;

**TypeName** - O nome da classe representada pelo *ObjectDataSource*;

**SelectParameters** - Possibilita obter uma colecção de parâmetros utilizado pelo método especificado na propriedade *SelectMethod*;

**InsertedMethod** - Afecta o nome do método que o controlo *ObjectDataSource* invoca no processo de inserção;

**InsertedParameters** - Obtém uma colecção de parâmetros usados pela propriedade *InsertMethod*.

### 4.3.2 *GridViewVotes*

Numa aplicação será necessário disponibilizar uma interface gráfica para visualização de itens e respectivas votações efectuadas pelo utilizador. O controlo *GridViewVotes* acrescenta esta nova funcionalidade, bastando que o cliente configure as propriedades que definem a fonte de dados dos itens e respectivas votações. Para visualização e edição das votações, o controlo cria internamente uma coluna com objectos do tipo *DropDownList*. O posicionamento da coluna de

votações pode ser definido através da propriedade *VotesColumnIndex*. Caso o cliente opte por não utilizar esta propriedade, a coluna de *votações* não ficará visível.

Este controlo, representado na figura 4.2, usa a técnica *AJAX*, sendo responsável por gerar o código *javascript* que invoca o *handler Vote*. É utilizada a biblioteca *JQuery* para facilitar a implementação.

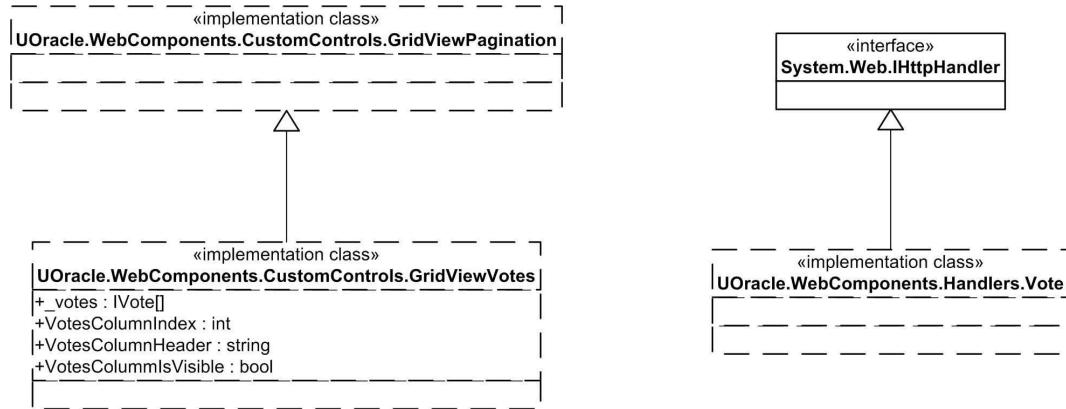


Figura 4.2: Diagrama de classes do componente *GridViewVotes*

Para além das propriedades disponibilizadas pela classe *GridViewPagination*, definem-se as propriedades intrínsecas de *GridViewVotes*:

***Votes*** - Afectar ou obter *IVote[]*, isto é, as votações do utilizador. Caso seja o cliente a fornecer estes dados, fica responsável por garantir a sua ordenação (pelo identificador único do item), uma vez que, internamente, o sistema realiza uma pesquisa dicotómica sobre as votações;

***VotesColumnIndex*** - O índice da coluna do objecto *GridView* onde se encontra o valor correspondente à votação;

***VotesColumnHeader*** - O texto a apresentar no cabeçalho da coluna de votações;

***VotesColumnIsVisible*** - A visibilidade da coluna de votações.

### 4.3.3 *GridViewPredictions*

Este componente baseia-se no *custom control* *GridViewVotes*, acrescentando-lhe uma coluna que indica a predição correspondente à associação utilizador/item. A configuração da coluna de predição deve ser feita através da propriedade *Predictions* no evento *OnPreRender*.

O controlo contém a propriedade *PredictionsColumnIndex* que possibilita configurar o posicionamento da coluna de predições. A omissão na configuração desta propriedade, implica a não visualização da coluna de predições.

Na figura 4.3 está ilustrado a hierarquia de classes do componente *DcGridViewPredictions*.

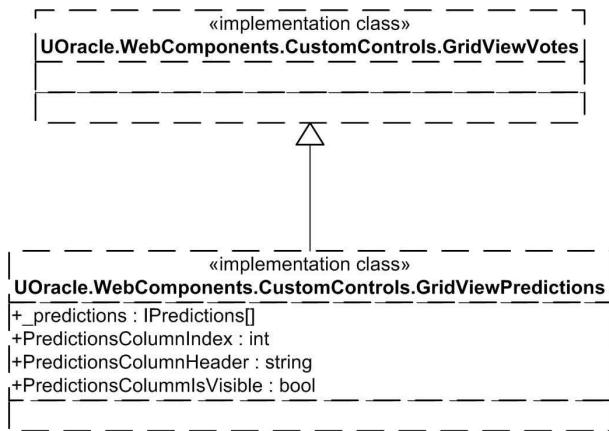


Figura 4.3: Diagrama de classes do componente *DcGridViewPredictions*

Definem-se as propriedades intrínsecas de *GridViewPredictions*:

**Predictions** - Afectar ou obter *IPrediction[]*, isto é, as previsões para o utilizador. Caso seja o cliente a fornecer estes dados, fica responsável por garantir a sua ordenação (pelo identificador único do item), uma vez que, internamente, o sistema realiza uma pesquisa dicotómica sobre as votações;

**PredictionsColumnIndex** - O índice da coluna do objecto *GridView* onde se encontra o valor correspondente à previsão;

**PredictionsColumnHeader** - O texto a apresentar no cabeçalho da coluna de previsões;

**PredictionsColumnIsVisible** - A visibilidade da coluna de previsões.

#### 4.3.4 WebControlItemTypeCategories

Este componente, dado um tipo, apresenta as categorias associadas. Esta funcionalidade pode ser utilizada pelo cliente na interface gráfica dos processos de pesquisa, inserção de itens, etc.

O componente, ilustrado na figura 4.4, utiliza a técnica *AJAX* para carregar as categorias associadas ao tipo seleccionado, sendo o *handler Categories* responsável por receber na *QueryString* o tipo e retornar um conjunto de categorias no formato *Json*.

Definem-se as propriedades intrínsecas de *WebControlItemTypeCategories*:

**ItemTypes** - Possibilita afectar ou obter um conjunto de tipos (*IItemType*);

**SelectedItemType** - Para obter o tipo seleccionado na *DropDownList* que lista os tipos;

**SelectedItemCategories** - Para obter as categorias seleccionadas na *DropDownList* que lista as categorias.

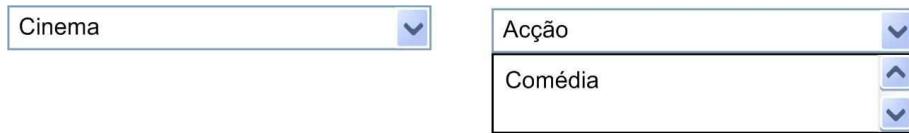


Figura 4.4: Componente *WebControlItemTypeCategories*

# Capítulo 5

## Protótipo da aplicação *web*

### 5.1 Introdução

Com o objectivo de testar o sistema num ambiente *web*, foi desenvolvido um protótipo de uma aplicação que visou o aperfeiçoamento do sistema, minimizando o esforço necessário ao desenvolvimento de aplicações por parte do cliente. Este exercício foi um contributo importante para a concepção do manual de utilização do sistema, em grande parte baseado na documentação deste protótipo. Assim, relega-se para a documentação do código fonte os pormenores de implementação, concentrando-se este capítulo nos seus aspectos estruturais.

### 5.2 Estrutura

Com a finalidade de demonstrar o processo de autenticação e autorização, foi pensada a seguinte organização de papéis:

**Utilizador Autenticado** - Vota em itens, podendo sugerir-los caso não existam no sistema;

**Moderador** - Cria itens, associando-lhes tipos e categorias. Valida a sugestão de itens efectuadas pelo *Utilizador Autenticado*;

**Administrador** - Cria tipos e categorias e gere as associações entre ambos. Cria *Moderadores*.

Para controlar o acesso a recursos (e.g., ficheiros *.aspx*), são definidas directórias. Na raiz de cada directória é incluído um ficheiro *web.config* onde é possível especificar os papéis com permissão de acesso. Apenas os utilizadores cujo papel coincide com a configuração efectuada podem aceder aos recursos presentes na respectiva directória. A figura 5.1 ilustra a organização da aplicação *web*.

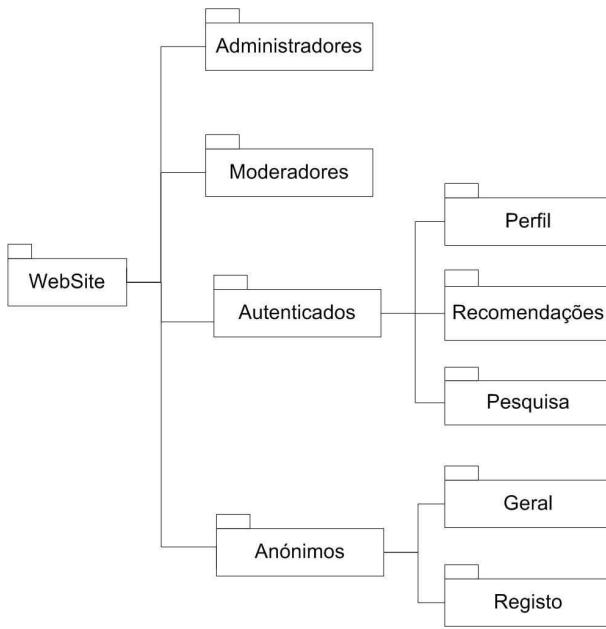


Figura 5.1: Organização da aplicação web

**Master pages.** Esta funcionalidade, disponibilizada pela *Framework .NET*, possibilita criar um modelo de apresentação global, que uniformiza o grafismo das páginas presentes na aplicação. Ao derivar de uma *master page*, a página herda a sua apresentação gráfica, sendo apenas definidos os conteúdos que a individualizam. Este conceito de herança facilita eventuais reestruturações, uma vez que as alterações efectuadas na *master page* se reflectem em todas as páginas derivadas. A figura 5.2 ilustra a organização das *master pages* utilizadas na concepção da aplicação web.

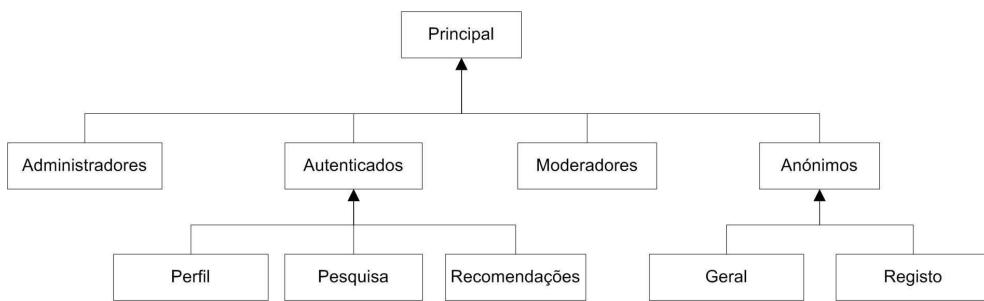


Figura 5.2: Organização das *master pages*

**Processo de autenticação.** No protótipo foi utilizado o modo de autenticação *Authentication Forms*. Este modo utiliza um objecto de contexto designado *authentication ticket*, criado após a validação das credenciais do utilizador. Este objecto é usualmente armazenado num *cookie*,

para que o utilizador permaneça autenticado em pedidos subsequentes ao servidor, isto é, durante o período de navegação no sítio, minimizando-se as ligações à base de dados. Foram utilizados os controlos *ASP.NET Login* para facilitar a interface gráfica na autenticação do utilizador, já que estes providenciam integração com o modo *Authentication Forms* e, conjuntamente com o *MembershipProvider*, automatizam o processo de autenticação.

**Componentes Web.** No protótipo, foram utilizados os componentes *web* apresentados no capítulo 4, para acesso às funcionalidades base do sistema.



# Capítulo 6

## Algoritmo de recomendação

### 6.1 Introdução

Com o objectivo de testar a capacidade do sistema na integração do algoritmo de recomendação, foi implementado um algoritmo de *Collaborative Filtering* (vertente *user-based*) [2], que aproxima utilizadores com perfis semelhantes, gerando um modelo sobre o qual é possível extrair recomendações.

Segue-se a enumeração dos passos do algoritmo de recomendação implementado, cuja leitura pode ser acompanhada com a ilustração presente na figura 6.1.

1. É construída a Matriz de Interacção entre utilizadores e itens ( $MI$ ), que representa os perfis dos utilizadores. Cada um dos vectores linha desta matriz representa as votações em itens de um utilizador;
2. É produzida a Matriz de Semelhança entre utilizadores ( $MS$ ), resultante da aplicação de uma medida de semelhança entre os vectores linha da  $MI$  (e.g., Co-Seno). Como pode ser observado na figura 6.1, a  $MS$  é simétrica ( $MI_{ij}=MI_{ji}$ ) e os elementos da sua diagonal principal têm o valor 1 (a semelhança de um utilizador com ele próprio);
3. É calculada a Matriz de Predições ( $MP$ ), que representa a previsão das votações dos utilizadores em itens. A  $MP$  é obtida através do produto matricial entre  $MS$  e  $MI$ , agregando as semelhanças entre utilizadores e as votações atribuídas. Salienta-se que, quanto maior for a semelhança entre dois utilizadores, maior será a influência das votações entre eles.

Perfil do utilizador				
	I1	I2	I1	I2
U1	2	3	0	0
U2	3	0	5	0
U3	0	3	0	5

Matriz Interacção

	U1	U2	U3
U1	1,0000	0,2854	0,4281
U2	0,2854	1,0000	0
U3	0,4281	0	1,0000

Matriz Semelhança

	I1	I2	I1	I2
U1	2,8562	4,2843	1,4270	2,1404
U2	3,5708	0,8562	5,0000	0
U3	0,8562	4,2843	0	5,0000

Matriz Predições

Figura 6.1: Matrizes envolvidas no cálculo do algoritmo

## 6.2 Implementação

Esta secção descreve a implementação do algoritmo, as estruturas de dados auxiliares que lhe dão suporte e as técnicas que melhoram o seu desempenho.

### 6.2.1 Contrato com o cliente

Tal como definido em 3.4.6, *assembly* resultante da compilação do código fonte é importado para o *SQL Server*, sendo mapeado o método estático que representa o procedimento armazenado *SpCreateModel*.

**Acesso a dados.** A interacção directa entre o algoritmo e o *SGBD* tem como objectivo maximizar a eficiência na criação do *modelo* de recomendação. Para que o algoritmo possa interagir directamente com os *perfis* sem comprometer a aplicação, é efectuada a replicação da tabela *Vote*, documentada em 3.4.8, para o acesso exclusivo do algoritmo.

### 6.2.2 Matriz de interacção

Pressupõe-se que o sistema atinja um número elevado de utilizadores e de itens catalogados e, visto que, as votações efectuadas pelos utilizadores são reduzidas face ao universo de itens,

a *MI* tende a ser esparsa<sup>1</sup>. Deste modo, é necessário utilizar uma estrutura que minimize o espaço ocupado em memória, armazenando apenas as votações efectuadas pelos utilizadores e que, simultaneamente, não comprometa a eficiência do sistema.

Como solução, é idealizada a estrutura *Elem*, representando a votação de um utilizador num item. As votações são armazenadas num *array* de objectos deste tipo, designado de *Array Elem*. A definição da estrutura *Elem* está patente na listagem 6.1.

```

1 public struct Elem<T>{
2     public int Index { get; set; }
3     public T Value { get; set; }
4 }
```

Listagem 6.1: Estrutura de dados *Elem*

O campo *Value* e *Index* do *Array Elem*, armazenam o valor e o **índice relativo** da votação. O índice relativo corresponde à posição onde a votação se encontra na *MI*. Para uma melhor compreensão, a figura 6.2 exemplifica uma possível matriz de interacção e os respectivos índices relativos das votações.

O campo *index* de *Elem* possibilita a indexação às linhas e colunas da *MI*. Para obtenção do índice da linha correspondente (utilizador), procede-se à divisão inteira do seu valor pelo número de colunas, isto é, pelo número total de itens. O índice da coluna (item), é obtido através do resto da divisão inteira do valor de *index* pelo número de colunas.

	I1	I2	I3	I4
U1	2	3	0	0
U2	3	0	5	0
U3	0	3	0	5

	I1	I2	I3	I4
U1	1	2	3	4
U2	5	6	7	8
U3	9	10	11	14

Figura 6.2: Matriz de interacção (à esq.) e respectivos índices relativos (à dta.)

Na figura 6.3 é apresentada a organização das votações dos utilizadores, na representação abstracta da *MI*, *Array Elem*. Chama-se a atenção para o número de elementos do *array*, que corresponde ao número total de votações dos utilizadores.

Elem(1,2)	Elem(2,3)	Elem(5,3)	Elem(7,5)	Elem(10,3)	Elem(12,5)
-----------	-----------	-----------	-----------	------------	------------

Figura 6.3: Organização da Matriz de Interacção no *Array Elem*

<sup>1</sup>Neste caso, com primazia de valores nulos.

Para dar suporte à consulta de votações no *Array Elem*, melhorando a eficiência do cálculo de previsões, é definido um *array* de elementos do tipo **Info**, designado *Array Info*. A estrutura *Info* está definida na listagem 6.2.

```

1 public class Info{
2     public int Idx { get; set; }          // user's first vote
3     public int NrOfElements { get; set; } //nr of votes
4     public float Norm { get; set; }       //euclidean norm
5     ...
6 }
```

Listagem 6.2: Estrutura de dados *Info*

Os elementos do *Array Info* armazenam a informação sobre as votações de um utilizador. Assim, por cada utilizador contemplado no cálculo do modelo de previsões, é criado um elemento *Info* preenchido com: o índice onde se encontra a sua primeira votação no *Array Elem*; o número de votações; o resultado do cálculo da norma euclidiana (utilizado para o cálculo da semelhança entre utilizadores em 6.2.3). Deste modo, a indexação a uma votação pode ser efectuada de uma forma simples, ilustrada na figura 6.4.

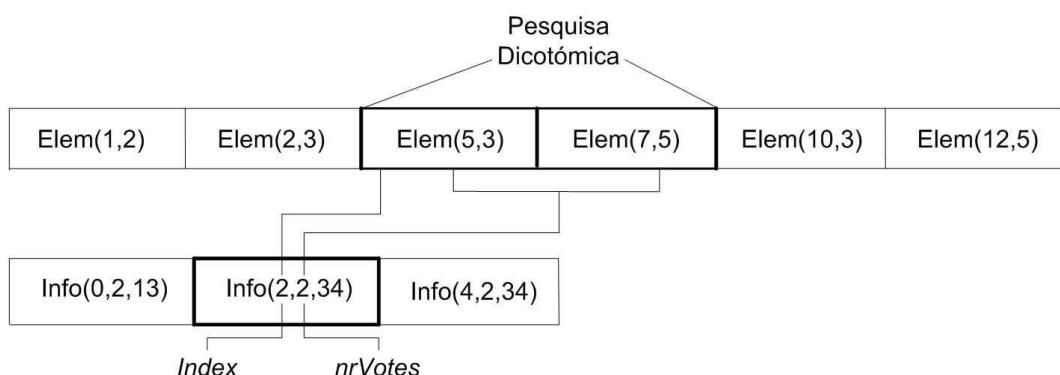


Figura 6.4: Pesquisa de uma votação no *Array Elem*

Para melhor compreensão da forma como é utilizado o *Array Elem*, explicam-se os passos necessários para obter a votação efectuada pelo utilizador nº2 no item nº3.

1. É indexado o índice 1 do *Array Info*<sup>2</sup>, de forma a saber onde se encontram as votações do utilizador, restringindo-se a pesquisa a um subconjunto de dois elementos do *Array Elem*.
2. É efectuada a pesquisa dicotómica nos elementos restantes.

<sup>2</sup>A informação sobre as votações do utilizador nº1 encontram-se no índice 0.

### 6.2.3 Matriz de semelhança

A matriz de semelhança entre utilizadores é calculada através da *MI*, aplicando uma medida de semelhança entre os vectores de linha (perfil dos utilizadores). A medida de semelhança, mede o ângulo entre os vectores dos utilizadores. A semelhança entre utilizadores é máxima quando os vectores que os representam são colineares, e nula, quando os vectores são ortogonais. A figura 6.5 ilustra o perfil de três utilizadores contemplando apenas dois itens.

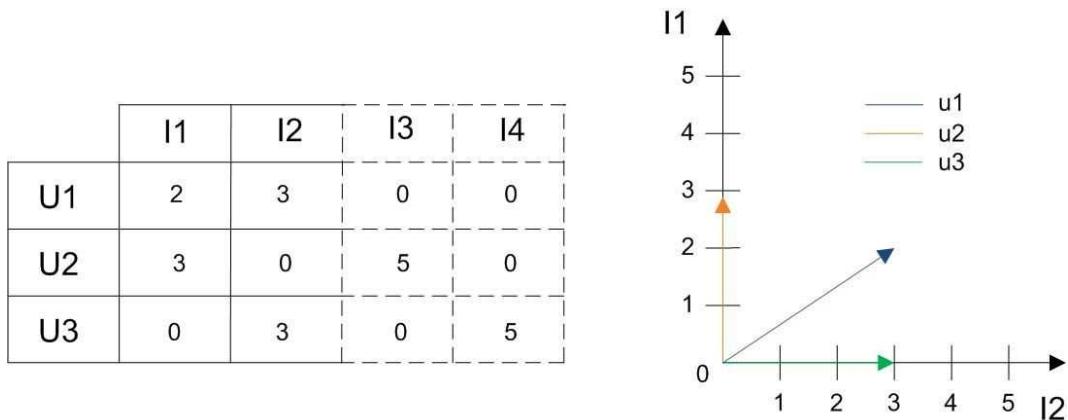


Figura 6.5: Representação em vectores do perfil de utilizadores

Na figura 6.5, os eixos do referencial correspondem aos itens, e a norma do vector é obtida através do valor das votações. Nesta implementação, é utilizado o co-seno como medida de semelhança entre vectores. A imagem 6.6 ilustra o cálculo da matriz de semelhança.

	U1	U2	U3	
U1	$\cos(\overrightarrow{u1}, \overrightarrow{u1})$	$\cos(\overrightarrow{u1}, \overrightarrow{u2})$	$\cos(\overrightarrow{u1}, \overrightarrow{u3})$	1,0000
U2	$\cos(\overrightarrow{u2}, \overrightarrow{u1})$	$\cos(\overrightarrow{u2}, \overrightarrow{u2})$	$\cos(\overrightarrow{u2}, \overrightarrow{u3})$	0,2854
U3	$\cos(\overrightarrow{u3}, \overrightarrow{u1})$	$\cos(\overrightarrow{u3}, \overrightarrow{u2})$	$\cos(\overrightarrow{u3}, \overrightarrow{u3})$	0,4281

	U1	U2	U3
U1	1,0000	0,2854	0,4281
U2	0,2854	1,0000	0
U3	0,4281	0	1,0000

Figura 6.6: Cálculo da Matriz de Semelhança

Observa-se que a semelhança entre um utilizador e ele próprio é máxima, e que o utilizador nº1 é mais semelhante com o nº2 do que com o nº3.

O facto da matriz ser simétrica e o valor dos elementos da sua diagonal principal serem 1, possibilita minimizar o espaço em memória, armazenando-se apenas o valor do cálculo dos elementos da diagonal superior, e reduzir o tempo de processamento, já que  $a_{ij}=a_{ji}$  e  $a_{ii}=1$ .

### 6.2.4 Matriz de predições

É obtida através do produto matricial entre a matriz de semelhança e a matriz de interacção e, posteriormente, da normalização dos valores resultantes. As predições são guardadas na tabela *Prediction*, presente no *SGBD*.

**Grau de confiança.** Este valor indica quantas votações foram consideradas na geração de uma previsão, o que acontece apenas quando dois utilizadores apresentam alguma semelhança entre si.

**Normalização das predições.** Os valores obtidos no produto matricial são normalizados, contextualizando-os ao utilizador a que pertencem, item, escala de valores das votações<sup>3</sup> e distribuição pelas classes de pontuação existentes. Segue-se uma descrição mais detalhada do processo de normalização:

- A contextualização relativa às predições do utilizador, é conseguida dividindo cada previsão pelo somatório das suas predições;
- A contextualização da previsão face ao item a que pertence é conseguida dividindo-a pelo número de votações contabilizadas na sua geração (*nrVotes*). Este enquadramento é necessário visto que: se o item *x* for mais votado que o item *y*, ainda que o valor das votações de *x* seja inferior face às de *y*, a previsão de *x* terá um valor superior;
- A contextualização face às possíveis votações é conseguida aplicando a regra de três simples. São conhecidas três variáveis da mesma grandeza e unidade: a previsão a normalizar, o valor máximo das predições do utilizador e o valor máximo de votação possível;
- Estima-se que o valor de previsões de graus mais elevados sejam reduzidos, assim, é definido qual o valor mínimo para que a previsão pertença a uma classe. Dito de outro modo, as previsões são distribuídas uniformemente pelos valores de pontuação possíveis na acção de voto.

As imagens 6.7 e 6.8 ilustram a matriz de previsões antes e após o processo de normalização.

Interpretação de resultados do resultado da imagem 6.8:

**Utilizador nº1.** Assemelha-se mais com o utilizador nº3 do que com o nº2, o que justifica o facto de o item nº4 ser mais recomendado que o item nº3 para o utilizador nº1.

---

<sup>3</sup>Neste caso de 1 a 5

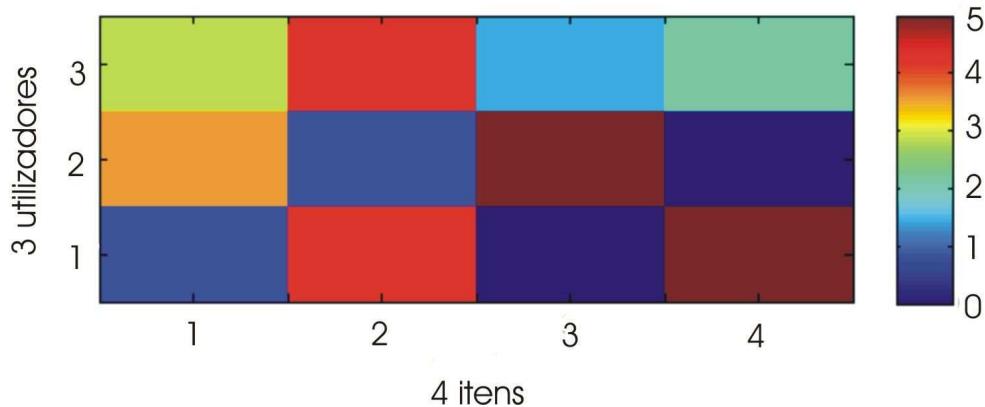


Figura 6.7: Matriz de Predições antes da normalização

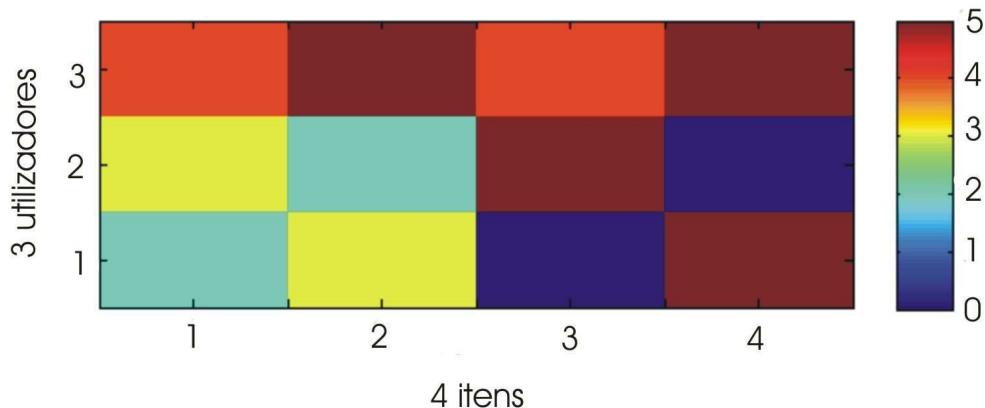


Figura 6.8: Matriz de Predições após a normalização

**Utilizador nº2.** Não tem semelhança com o utilizador nº3, logo, as suas votações não polarizam as predições do item nº2 e nº4. As votações do utilizador nº1 influenciaram as predições para os itens nº1 e nº2.

**Utilizador nº3.** Não tem semelhança com o utilizador nº2, O utilizador nº1 influenciou as predições para o item nº1 e nº 2.

### 6.2.5 Persistência do modelo de recomendação

No processamento das predições é aproveitada a sintaxe disponível no *SQL Server 2008* para inserção de um conjunto de previsões numa única transacção, até um máximo permitido de mil registos. O algoritmo de recomendação, após o cálculo do modelo de predições, invoca a *stored procedure SpUpdateModel* que, através de uma transacção distribuída, actualiza o modelo de recomendação.

### 6.2.6 Testes de desempenho

A tabela 6.1 apresenta os testes de desempenho efectuados ao algoritmo de recomendação. O conjunto de dados utilizado para testes (votações dos utilizadores) são dados reais pertencentes ao projecto *MovieLens*. Efectuaram-se testes até 1.000 utilizadores, 155.177 votações em 3883 itens. Na tabela 6.1 podem observar-se o número de predições de cada modelo calculado, bem como o tempo de processamento.

Utilizadores	Votações	Predições	Tempo
0	0	0	0
1	53	53	1"
5	452	1.925	2"
10	1.200	7.860	5"
20	2.524	23.154	12"
30	4.096	44.566	22"
40	5.678	68.078	34"
50	7.260	93.556	49"
100	12.976	230.014	2'0"
200	29.415	565.040	5'24"
500	73.871	1.583.516	19'04"
1000	155.177	3.377.752	51'14"

Tabela 6.1: Testes de desempenho do algoritmo

# Capítulo 7

## Conclusões

Neste capítulo faz-se um resumo dos resultados obtidos na implementação do sistema de recomendações *UOracle* e são apontados alguns pontos de extensão para trabalho futuro.

### 7.1 Resultados

O projecto desenvolvido, com a designação de *UOracle*, simplifica o desenvolvimento de aplicações de recomendação, possibilitando o registo e autenticação de utilizadores, e o controlo no acesso a recursos através de um processo de autorização baseado em papéis (*role-based*). Contempla a criação de perfis de utilizadores e integra, de forma fácil, algoritmos de recomendação. Com base no modelo gerado pelo algoritmo de recomendação integrado no sistema, responde às solicitações de recomendação efectuadas pelos utilizadores. Como exemplo de utilização, é facultado o protótipo de uma aplicação *web* que explora as funcionalidades principais do sistema.

Foram concretizados todos os requisitos mínimos constantes da proposta. Superando as expectativas iniciais, foi também concretizado o requisito opcional, que consistia na implementação de um algoritmo de recomendação baseado em *Collaborative Filtering*, documentado no capítulo 6. Após cumpridos os objectivos propostos, foram idealizadas formas de valorizar este projecto. No capítulo 4 está documentado o desenvolvimento dos componentes .NET que facilitam a criação de aplicações, minimizando o esforço do cliente na concepção da interface gráfica que dá acesso às funções principais do sistema. Nas funcionalidades adicionais implementadas, destaca-se a concepção do sistema automático de triagem *BingItemSuggestion*, documentado em 3.4.7, que possibilita analisar o grau de correcção das sugestões de itens submetidas por utilizadores.

### 7.1.1 Dificuldades identificadas

Numa fase inicial, foi necessário delimitar a abrangência deste projecto, para que pudesse ser concretizado no prazo estabelecido. Por esse motivo, decidiu-se pela utilização da *Entity Framework* na criação da camada de acesso a dados. O facto de esta tecnologia não estar contemplada no programa curricular desta licenciatura, levou a uma incursão com pouco tempo de preparação e a sucessivos ajustes na sua utilização.

Um problema constatado numa fase adiantada do projecto, foi a impossibilidade de se partilhar uma ligação à base de dados entre a *Entity Framework* e o *SqlMembershipProvider*, uma vez que o último não devolve a ligação para que possa ser partilhada. Como consequência, transacções que envolvam simultaneamente a camada de acesso a dados e o *provider*, são promovidas a transacções distribuídas apesar de serem efectuadas na mesma máquina. Este problema pode ser constatado em 3.4.4.

A implementação do algoritmo de recomendação, documentada no capítulo 6, foi uma das tarefas mais complicadas de concretizar. A quantidade elevada de dados originou problemas de memória e de processamento. Numa primeira implementação, foi utilizada como estrutura de dados um *array* bidimensional que armazenava os valores das associações utilizadores/itens. Esta estrutura, embora facilitasse a indexação, tinha como inconveniente um acentuado desperdício de memória nas votações nulas, isto é, itens não votados pelos utilizadores. Na implementação final, recorreu-se a uma estrutura de dados que possibilita armazenar apenas as votações efectivas dos utilizadores. Foi possível calcular o modelo de recomendação para 155.177 votações, de 1.000 utilizadores em 3.883 itens, em aproximadamente 51 minutos, gerando um total de 3.377.752 predições.

## 7.2 Considerações para trabalho futuro

Num trabalho desta natureza, existem sempre alguns aspectos que mereceriam maior atenção que não foi possível dedicar-lhes por falta de tempo. No caso concreto deste projecto, destacam-se como merecendo maior desenvolvimento futuro:

**A camada de acesso a dados.** Como ponto de extensão deste trabalho, considera-se a implementação (de raiz) de uma camada de acesso a dados que possibilite o controlo absoluto das operações *CRUD*, maximizando a eficiência do sistema. A inclusão do conceito de sessão possibilitaria partilhar, entre objectos, a mesma ligação à base de dados, anulando o custo associado ao estabelecimento da mesma.

**A segurança.** Devido à natureza académica deste trabalho, foram negligenciados aspectos relacionados com a segurança do sistema, que num contexto real teriam de ser considerados. Um exemplo é o acesso directo às tabelas do *SGBD*. Como possível solução para este problema, poderiam ser criados procedimentos armazenados que, associados às operações *CRUD*, controlariam o processo de inserção e actualização de dados. O acesso directo às tabelas seria restringido através de permissões.

**O algoritmos de recomendação.** No desenvolvimento do algoritmo de recomendação foram identificados como pontos críticos, o problema das matrizes esparsas e a eficiência do cálculo matricial. Com o acentuado desenvolvimento das placas gráficas, suportando processamento paralelo com vários núcleos, e o aparecimento de *APIs* que possibilitam ao programador utilizar esta capacidade de processamento, surge uma oportunidade para melhorar o desempenho no cálculo do modelo de recomendações. A *API CUDA* [16], disponibilizada para interacção com a última geração de placas gráficas da empresa *NVIDIA*, trata eficientemente o problema de armazenamento e do cálculo matricial de matrizes esparsas. A sua utilização seria um contributo importante para que o sistema pudesse ser utilizado num cenário real.



# Apêndice A

## Documentação auxiliar

### A.1 Modelo relacional

#### Entidade UOracleRole

- **UOracleRole( uoracleRoleName )**
  - Chave Candidata = { uoracleRoleName }
  - Chave Primária = { uoracleRoleName }
- Esquema de Relação Resultante:

-**UOracleRole( uoracleRoleName )**

#### Entidade UOracleUser

- **UOracleUser( uoracleUserId )**
  - Chave Candidata = { uoracleUserId }
  - Chave Primária = { uoracleUserId }
- Esquema de Relação Resultante:

-**UOracleUser( uoracleUserId )**

#### Entidade UOracleUsersInRoles

- **UOracleUsersInRoles( uoracleUserId, uoracleRoleName )**
  - Chave Candidata = { uoracleUserId, uoracleRoleName }

- Chave Estrangeira = { uoracleUserId }, { uoracleRoleName }
- Chave Primária = { uoracleUserId, uoracleRoleName }
- Esquema de Relação Resultante:

**-UOracleUsersInRoles( uoracleUserId, uoracleRoleName )**

### **Entidade ItemType**

- **ItemType(itemType )**
  - Chave Candidata = { itemType }
  - Chave Primária = { itemType }
- Esquema de Relação Resultante:

**-ItemType( itemType )**

### **Entidade ItemCategory**

- **ItemCategory(itemCategory )**
  - Chave Candidata = { itemCategory }
  - Chave Primária = { itemCategory }
- Esquema de Relação Resultante:

**-ItemCategory( itemCategory )**

### **Entidade ItemTypeItemCategory**

- **ItemTypeItemCategory(itemType, itemCategory )**
  - Chave Candidata = { itemType, itemCategory }
  - Chave Estrangeira = { itemType }, { itemCategory }
  - Chave Primária = { itemType, itemCategory }
- Esquema de Relação Resultante:

**-ItemTypeItemCategory( itemType, itemCategory )**

### Entidade Item

- **Item(itemId, itemType, title )**
  - Chave Candidata = { itemId }
  - Chave Estrangeira = { itemType }
  - Chave Primária = { itemId }
- Esquema de Relação Resultante:

**-Item( itemId, itemType, title )**

### Entidade SuggestedItem

- **SuggestedItem(id, itemType, title, isvalidated )**
  - Chave Candidata = { id }
  - Chave Estrangeira = { uoracleUserId }
  - Chave Estrangeira = { itemType }
  - Chave Primária = { id }
- Esquema de Relação Resultante:

**-SuggestedItem( id, itemType, title, isvalidated )**

### Entidade Item\_ItemTypeItemCategory

- **Item\_ItemTypeItemCategory(itemId, itemType, itemCategory )**
  - Chave Candidata = { itemId, itemType, itemCategory }
  - Chave Estrangeira = { itemId }, { itemType }, { itemCategory }
  - Chave Primária = { itemId, itemType, itemCategory }
- Esquema de Relação Resultante:

**-Item\_ItemTypeItemCategory( itemId, itemType, itemCategory )**

### Entidade Vote

- **Vote(uoracleUserId, itemId, score )**
  - Chave Candidata = { uoracleUserId, itemId }
  - Chave Estrangeira = { uoracleUserId }, { itemId }

- Chave Primária = { uoracleUserId, itemId }
- Esquema de Relação Resultante:

**Vote( uoracleUserId, itemId, score )**

### **Entidade Prediction**

- **Prediction(uoracleUserId, itemId, prediction )**
  - Chave Candidata = { uoracleUserId, itemId }
  - Chave Estrangeira = { uoracleUserId }, { itemId }
  - Chave Primária = { uoracleUserId, itemId }
- Esquema de Relação Resultante:

**-Prediction( uoracleUserId, itemId, prediction )**

## A.2 Entity Framework EDM (Entity Data Model)

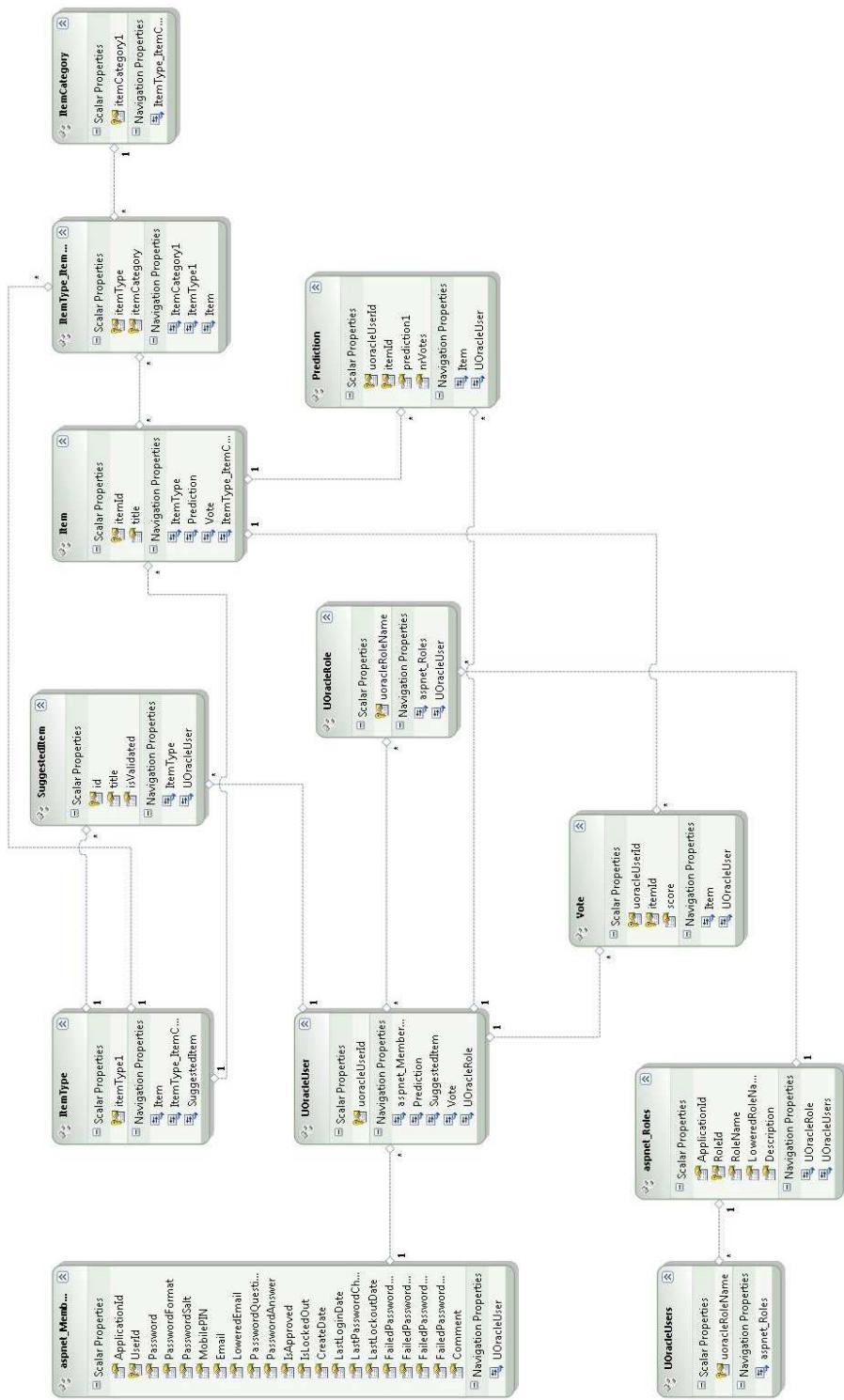


Figura A.1: *Entity Data Model* gerado pelo *Entity framework*

### A.3 Perspectiva geral do sistema

A hierarquia de classes representada na figura A.2, dá uma perspectiva geral do sistema de recomendações.

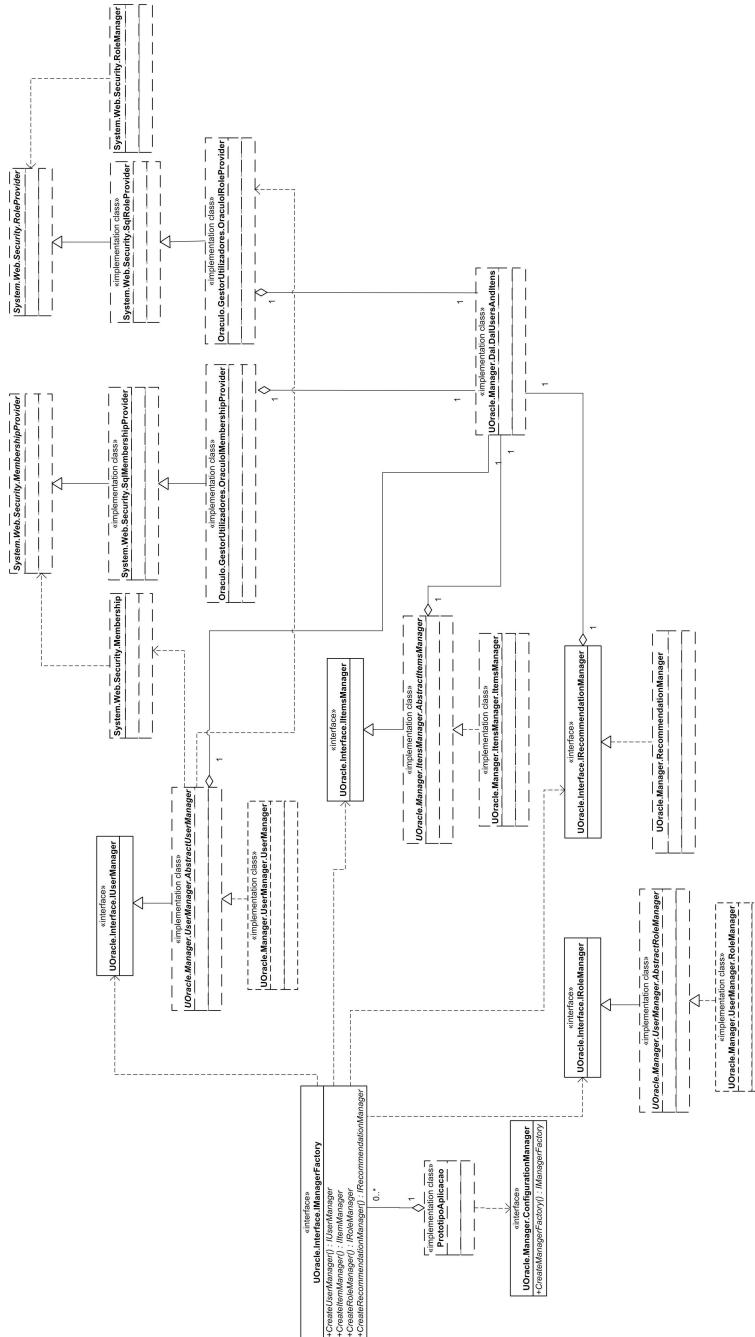


Figura A.2: Diagrama de classes do sistema de recomendações

## A.4 Diagrama de classes do algoritmo de recomendação

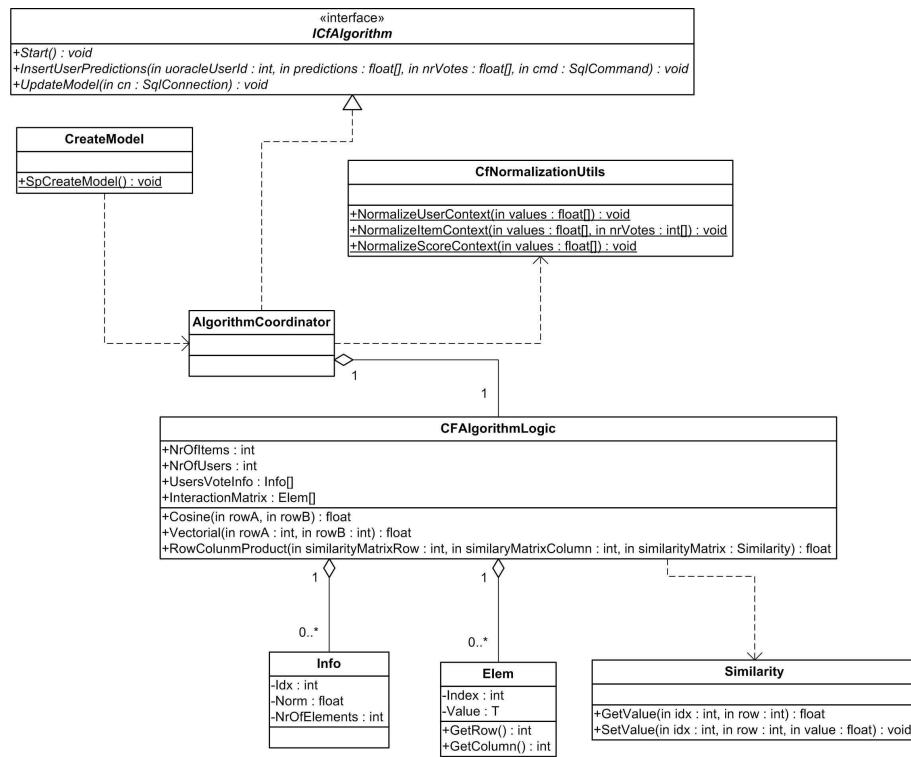


Figura A.3: Diagrama de classes do algoritmo de recomendação



# Referências

- [1] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl, *Analysis of Recommendation Algorithms for ECommerce*, University of Minnesota, Minnesota, USA, 2000 ([www.grouplens.org/papers/pdf/ec00.pdf](http://www.grouplens.org/papers/pdf/ec00.pdf)), visitado em 15 de Março, 2009.
- [2] Zan Huang, Daniel Zeng, and Hsinchun Chen, *A Comparative Study of Recommendation Algorithms in ECommerce Applications*, (<http://ai.arizona.edu/go/intranet/papers/comparative.ieeeis.pdf>), visitado em 18 de Março, 2009
- [3] Greg Linden, Brent Smith, and Jeremy York, *Amazon.com Recommendations - Item-to-Item Collaborative Filtering*, IEEE Computer Society, Janeiro-Fevereiro, 2003. (<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01167344>), visitado em 17 de Março, 2009.
- [4] <http://www.bing.com/developers/tou.aspx>, Microsoft, visitado em 15 de Agosto, 2009.
- [5] Walter Vieira, "Texto de Apoio à Cadeira de Sistemas de Informação", ISEL, 2007.
- [6] Julia Lerman , *Programming Entity Framework*, first Edition, O'Reilly , 12 Fevereiro, 2009.
- [7] Cay Horstmann, "Object-Oriented Design And Patterns, second edition", Wiley, 2006.
- [8] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Elements of Reusable Object-Oriented Software", Addison Wesley, 1995
- [9] Walter Vieira, *Integração do CLR no Sql Server 2005*, ISEL/DEETC, 2005.
- [10] Thomas Rizzo, *Pro SQL Server 2000*, Apress, 2005.
- [11] Peter DeBetta, *Introducing Microsoft SQL Server 2005 for Developers*, Microsoft Press, 2005.
- [12] Design Guidelines for Developing Class Libraries, <http://msdn.microsoft.com/en-us/library/ms229002.aspx>, visitado em 2 de Julho, 2009.
- [13] <http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx>, visitado em 16 de Abril, 2009.

- [14] <http://msdn.microsoft.com/en-us/library/aa479347.aspx>, visitado em 16 de Abril, 2009.
- [15] <http://msdn.microsoft.com/en-us/library/9a4kyhcx.aspx>, visitado em 16 de Abril, 2009.
- [16] Nathan Bell and Michael Garland, *Efficient Sparse Matrix-Vector Multiplication on CUDA*, NVIDIA Technical Report NVR-2008-004, 2008