# Relatório de Bases de Dados

Banana Store – A sua loja de música

Realizado por:

* Guilherme Reis de Micaelo Simões nº2008110352
* João Miguel Montenegro dos Santos Barroso Penetra nº2008111633
* Ricardo Jorge Rocha Bernardino nº2008114731

## Indice

## Introdução

Este projecto foi realizado no âmbito da cadeira de Bases de Dados com o intuito de aplicar os conhecimentos obtidos ao longo do semestre. Por ser um tema que interessava a todos os membros do grupo escolhemos o tema música, nomeadamente criação de uma loja de vendas. Por o nosso grupo ser constituído por 3 elementos, decidimos aumentar a complexidade do projecto elaborando uma loja online de vendas. Estas vendas incluem: álbuns, músicas em formato digital e merchandising dos artistas. A compra de álbuns e de merchandising funciona como um sistema normal de encomendas com entrega no domicílio. As músicas em formato digital são apenas uma simulação da vida real já que neste projecto não disponibilizamos qualquer música. É também importante frisar que o dinheiro não é real e os utilizadores podem carregar as suas contas com a quantidade de créditos que desejarem, sem qualquer restrição.

Para aumentar a utilidade do projecto, decidimos incluir na base de dados informações sobre os artistas que têm material disponível na nossa loja. Para facilitar a inserção dos dados na base de dados utilizámos a API do conhecido site *“last.fm”*. Assim, o administrador do site pode adicionar com facilidade material e informações dos artistas utilizando esta API que possui uma enorme quantidade de informação disponível sobre quase todos os artistas do mundo. Estas informações incluem descrições de bandas, músicas e álbuns e imagens. De notar, que apenas são guardados os *links* para as imagens e não as imagens em si.

Para aumentar a facilidade de encontrar artigos, o site possui um sistema de procura por artista, álbum, música e merchandising.

Para criar este projecto utilizámos a linguagem Ruby. O facto de esta ser uma linguagem nova para nós ainda contribuiu mais para nos empenharmos na elaboração do mesmo. Para a criação do servidor web utilizámos a framework Sinatra, uma framework web bastante simplista e que não depende de base de dados como acontece com outras frameworks bastante conhecidas como o Ruby on Rails e Django (Python).

## Definição do Produto

### - Visão do produto

Pretendemos construir uma aplicação de bases de dados para uma loja de música online. A aplicação do lado do administrador permitirá consultar estatísticas de vendas e também gerir o stock da loja, isto é, adicionar, remover e alterar a informação relativa aos produtos existentes. Da parte do utilizador comum, a aplicação possibilitará procurar, marcar para futura compra, encomendar e classificar produtos. Os produtos disponibilizados pela loja irão desde álbuns e *singles* a *merchandising* das bandas. O utilizador poderá fazer procuras por nome da música, álbum, artista, editora, classificação, género e por outros possíveis atributos.

Em relação aos requisitos de software e hardware tudo dependerá da complexidade da implementação da nossa aplicação. Se decidíssemos integrar o nosso trabalho na Web seria necessário ter um serviço de alojamento para a nossa base de dados e o cliente teria de ter acesso à internet. No entanto, dado que este projecto é meramente académico e as aplicações distribuídas não são o principal objectivo desta cadeira, poderíamos não desenvolver as características supramencionadas.

### - Relevância e interesse

Esta aplicação que pretendemos desenvolver é bastante actual e, como tal, tem bastante relevância no negócio da música de hoje em dia. Num mundo cada vez mais global e ligado, e onde a música continua a ter um papel muito importante na nossa sociedade, a incorporação de serviços de algumas redes sociais como o Twitter e Last.fm seria uma mais-valia para o nosso produto final. O uso das APIs destas redes é interessantíssimo e seria um factor distintivo do nosso projecto mas a complexidade da sua implementação pode dificultar a concretização do mesmo. De qualquer forma, o desenvolvimento do nosso software, mesmo sem a integração destas redes sociais, seria pertinente e útil para utilizar numa loja real.

## Plano de Projecto

|  |  |  |
| --- | --- | --- |
| **Nome da Tarefa** | **Início** | **Fim** |
|  |  |  |
| **Meta 1** | 08-10-2010 | 11-10-2010 |
| Definição do Produto e Plano de Projecto | 08-10-2010 | 11-10-2010 |
|  |  |  |
| **Meta 2** | 11-10-2010 | 15-11-2010 |
| Especificação funcional | 11-10-2010 | 15-11-2010 |
| Modelo de ER | 11-10-2010 | 15-11-2010 |
| Plano de Desenvolvimento de Código | 11-10-2010 | 15-11-2010 |
|  |  |  |
| **Meta 3** | 01-11-2010 | 03-01-2011 |
| Especificação técnica final | 01-11-2010 | 23-12-2010 |
| Desenvolvimento do Código | 01-11-2010 | 23-12-2010 |
| Plano final de desenvolvimento de Código | 01-12-2010 | 23-12-2010 |
| Relatório | 20-12-2010 | 03-01-2011 |

## Especificação funcional

### Estrutura da Base de Dados

Datas: 16-11-2010 até 28-11-2010

Inicialmente, iremos criar a estrutura da base de dados utilizando as funcionalidades do PowerDesigner. Vamos proceder também à definição das restrições e dependências entre tabelas. Para verificar que as restrições dos atributos e as dependências entre tabelas são os mais correctos, iremos testar as mesmas com vários testes.

### Definição do Design da Aplicação Web

Datas: 28-11-2010 até 2-12-2010

Apoiando-nos nos mockups que seguem em anexo, iremos proceder à criação de mais mockups de outras possívies funcionalidades. Esta fase permite facilitar o desenvolvimento do design do site.

### Implementação do Design

Datas: 3-12-2010 até 20-12-2010

Com alguns dos principais mockups já feitos, iremos começar a criar as páginas web com o design definido.

### Escrita do código das funcionalidades

Datas: 3-12-2010 até 28-12-2010

Nesta fase iremos desenvolver as principais funcionalidades da nossa loja online utilizando várias linguagens de programação. Pretendemos utilizar: SQL, PLSQL, Java/JSP ou Ruby/Sinatra.

### Integração do código com a Aplicação Web

Datas: 10-12-2010 até 3-1-2011

À medida que vamos escrevendo o código, iremos ligar esse mesmo código à interface da nossa aplicação web.

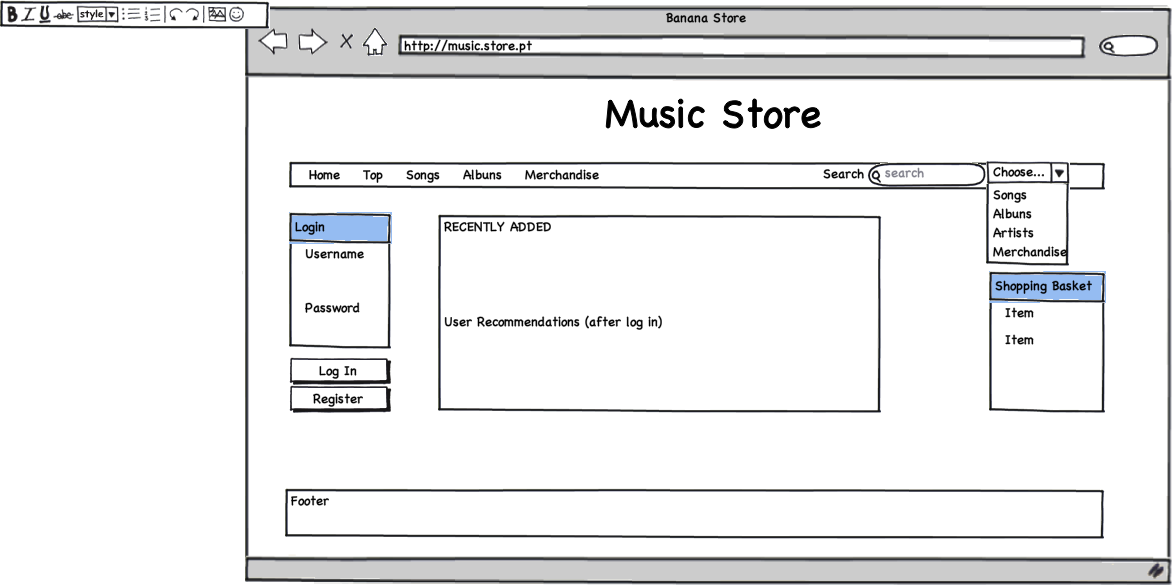


Ilustração - Página principal do site

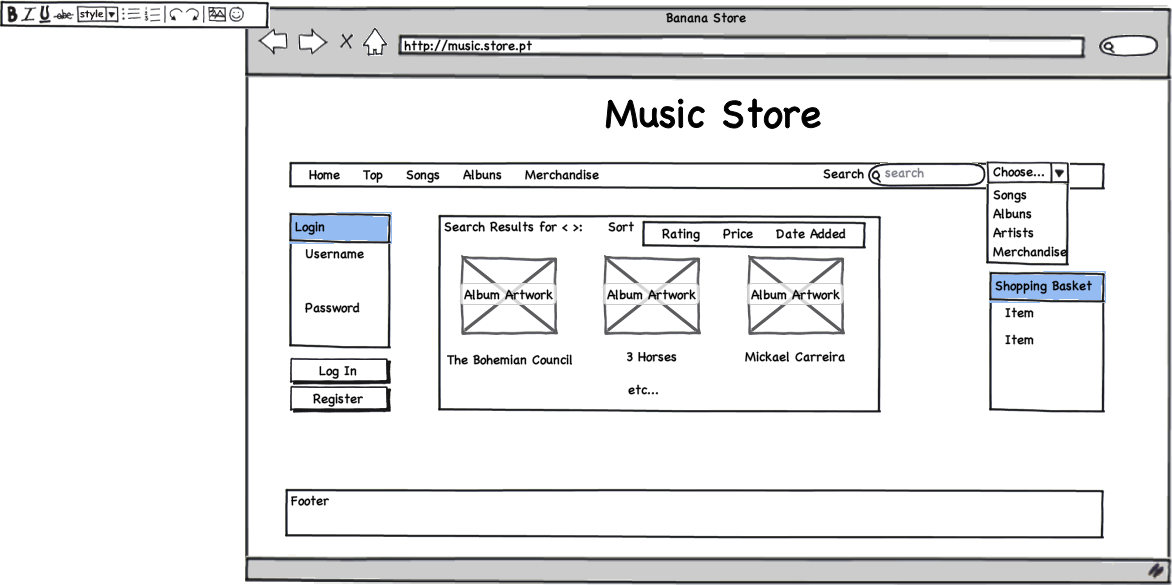


Ilustração - Página de Procura

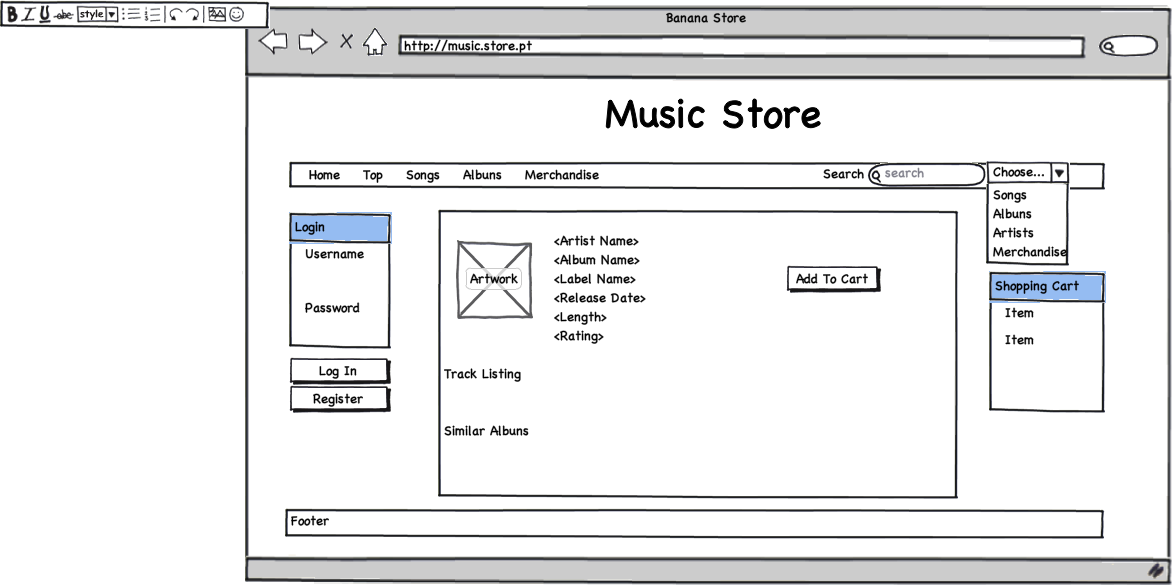


Ilustração - Página de um álbum

## C:\Users\João Penetra\Documents\My Dropbox\BD\Imgs\ER.PNGDiagrama ER

## Especificação técnica final

### Arquitectura da Aplicação Web

A nossa aplicação web é baseada no modelo MVC (model, view, controller), ou seja, a lógica da aplicação é isolada da interface gráfica do utilizador, permitindo desta forma um desenvolvimento, teste e manutenção independentes entre ambas as partes. No model, temos toda a lógica de manutenção dos dados da aplicação, que no nosso caso é assegurada pelo sistema de gestão de bases de dados Oracle. No view, é onde fica todo o código da interface gráfica da aplicação, que no nosso caso são ficheiros .ERB contendo códigos HTML, CSS, JavaScript e Ruby. O controller, recebe pedidos GET e POST referentes ao protocolo HTTP e inicia a resposta fazendo chamadas aos objectos na parte do model, caso seja necessário. Na nossa aplicação o código representativo do Controller está disponível no ficheiro "store.rb", em que são visíveis as diferentes routes disponiveis na aplicação. É aqui que podemos constatar a simpliciadade da framework sinatra, pois para cada route apenas temos que utilizar um dos verbos http (get, post, delete e put), a path desejada e as instruções a efectuar quando é feito um pedido áquela route. Outra particularidade interessante da framework é a ligação à base de dados ser específica da aplicação, e não ser criada por cada pedido, aumentando desta forma a performance comparativamente a PHP, por exemplo. Para estabelecer a ligação à mesma, utilizamos uma gem do ruby denominada oci8 especificamente destinada a bases de dados Oracle - http://ruby-oci8.rubyforge.org/en/.

### Definição de Classes

Depois de referir como estruturámos a nossa aplicação, é útil explicar onde é que estão definidas as funções que possibilitam o normal funcionamento do produto final.

#### Database.rb

Esta é a classe responsável pela interacção com a base de dados. Para estabelecer esta ligação utilizámos a interface Ruby-oci8. Assim, quando o servidor é ligado é executado o método “*initialize”* para estabelecer esse contacto com o Oracle. Em seguida, sempre que for necessário interagir com a base de dados podem ser chamados os métodos *“select”* ou *“execute”* consoante o utilizador pretende consultar alguma informação ou o administrador está a inserir ou alterar alguma informação.

#### Lastfm.rb

Como referido anteriormente, o administrador da loja tem a possibilidade de adicionar, editar e remover conteúdos (artistas, álbuns, músicas e merchandising). Para facilitar esse trabalho, criaram-se vários métodos disponíveis nesta classe. Esses métodos fazem uso da API Rest do site Last.fm (<http://www.last.fm/api>). Os métodos existentes são:

* addArtist()
* addAlbum()
* updateArtist()
* updateAlbum()

Todos estes métodos são chamados a partir de *“store.rb”* e apenas servem para extrair a informação da API do site. Concluída esta fase, são chamados métodos na classe *“Manage.rb”* que interage com a base de dados. Falemos então da referida classe.

#### Manage.rb

A informação que está na base de dados pode ser actualizada manualmente a qualquer altura pelo administrador. Foi necessário, por isso, criar uma classe independente da do Last.fm que executa procedimentos de introdução ou simples updates e delete na base de dados. Esses métodos têm nomes satisfatoriamente esclarecedores:

* addArtist() – adicionar um artista;
* addProduct() – Cada música, álbum ou merchandising descende de uma entidade produto como demonstrado no modelo ER;
* addAlbum() – adicionar um álbum;
* addSong() – adicionar uma música;
* addMerch() – adicionar uma peça de merchandising;
* addClient() – registar um cliente no sistema;
* addOrder() – registar uma venda;
* deleteArtist() – apagar um artista;
* deleteAlbum() – apagar um álbum;
* deleteSong() – apagar uma música;
* deleteMerch() – apagar uma peça de merchandising;
* deleteClient() – apagar um cliente do sistema.

#### Get.rb

Esta classe é responsável por executar *“queries”* à base de dados consoante a navegação dos utilizadores. Permite portanto obter todas as informações referentes aos artistas, álbuns, músicas e merchandising. Tem por isso métodos para ir buscar estas informações mas também outros que permitem formar os tops de vendas, adições recentes e listas de conteúdos.

#### Search.rb

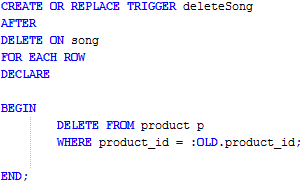
Classe que possui métodos para devolver informações das pesquisas efectuadas. Para facilitar ao utilizador a obtenção da informação pretendida, existem procuras pelos seguintes parâmetros: artistas, álbuns, músicas, merchandising e género. Todos estes métodos retornam as informações consoante o filtro introduzido pelo utilizador.

### Triggers

Os *“triggers”* são uma ferramenta muito importante para manter a consistência dos dados de uma base de dados. Vamos por isso explicar que *“triggers”* foram implementados no nosso projecto.

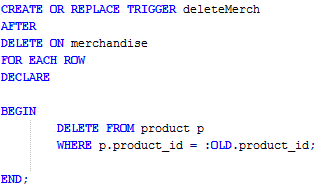
#### deleteSong

Este *“trigger”* é accionado depois de ser feito um delete na tabela das músicas. É utilizado para apagar o produto respectivo à música.



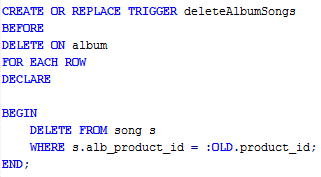
#### deleteMerch

Este *“trigger”* é accionado depois de ser feito um delete na tabela do merchandising. É utilizado para apagar o produto respectivo à peça eliminada.



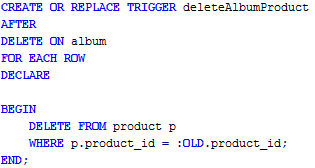
#### deleteAlbumSongs

Este *“trigger”* é accionado antes de ser feito um delete na tabela dos albums. Como um álbum tem sempre um conjunto de músicas que o referenciam, é necessário apagar essas músicas para que seja possível proceder ao delete do album.



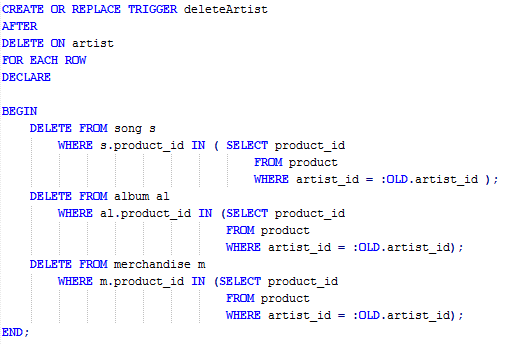
#### deleteAlbumProduct

Tal como existe um *“trigger”* respectivo ao delete do produto referente ao álbum e a uma música, este é responsável por apagar o produto que corresponde ao álbum. Este é accionado depois de feito o delete na tabela dos álbuns.



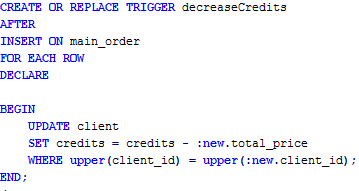
#### deleteArtist

*“Trigger”* accionado antes de ser feito um delete na tabela dos artistas. É responsável por apagar todas as músicas, álbuns e merchandising referente a esse artista.



#### decreaseCredits

Este *“trigger”* é accionado depois de ser adicionada à base de dados uma nova compra (main\_order). É responsável por diminuir os créditos na conta do utilizador que fez a utilizando o preço total da compra.



#### sellItem

*“Trigger”* accionado por cada item que é vendido. É responsável por actualizar a tabela dos produtos diminuindo o stock existente e aumentado o número de vendas consoante a quantidade desse produto que o cliente comprou. Torna possível a existência de tops de vendas e controlo de stock.

