universidade de aveiro

Novas Tecnologias da Comunicação

Laboratório Multimédia 4

Mini-Projeto Individual

LP

https://labmm.clients.ua.pt/deca_20L4/deca_20L4_03/MP/

aluno@aluno.com — 12345678

Cartão de Crédito (teste) — 4242 4242 4242 4242 (data futura, 3 números aleatórios)

Simão Bentes (97761)

1. Introdução

No sentido de aprofundar os conhecimentos adquiridos na cadeira de Laboratório Multimédia 4, decidi desenvolver um projeto de trabalho ambicioso que simultaneamente obedecesse e desafiasse alguns dos conceitos que temos vindo a desconstruir na unidade curricular. A plataforma que desenvolvi está intrinsecamente ligada à edição fonográfica de música em suporte vinil. Os vinis sempre “andaram lá por casa” e exerceram em mim um certo fascínio que quis agora integrar como tema principal do mini-projeto. Uma indústria que teve o seu auge nos anos 70-80, o desaparecimento nos anos 90 e que agora estão definitivamente de volta.

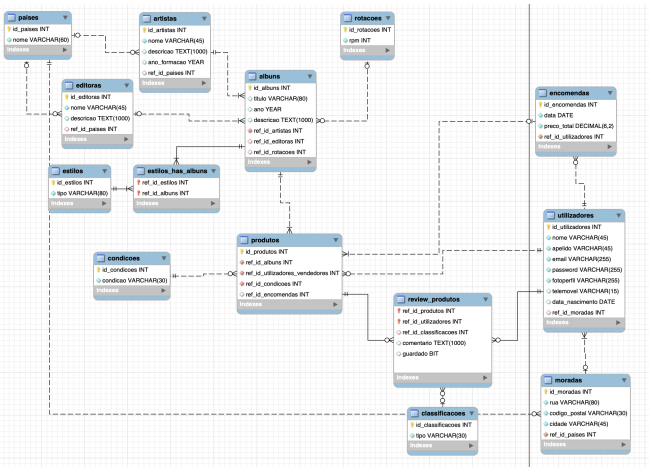
O objetivo foi assegurar uma linha de motivação criativa para o seu desenvolvimento e poder mostrar um pouco da minha identidade. Assim, decidi dar à aplicação o nome de “LP”. A plataforma permite vender e comprar álbuns, classificar e fazer reviews.

Apesar de ter sido uma recomendação dos docentes, optei por não recorrer a um tema de Bootstrap e utilizar antes o meu projeto final de Laboratório Multimédia 2 como base para a implementação do desafio. Senti que este, para além de ter uma linha de design que aprecio, era suficientemente sólido para suportar as exigências de um projeto como este.

2. Modelo ER e narrativa de utilização

Antes de partir para a criação do modelo relacional, criei uma lista de requisitos que a minha aplicação tinha que respeitar. Esta fase do projeto foi particularmente desafiante, na medida em que estava a propor criar algo, sem ter bem a noção do tempo e conhecimento que teria para implementar tudo aquilo que o modelo ER propunha. Irei primeiro abordar a evolução do modelo relacional ao longo do projeto e, posteriormente, explorarei as suas características.

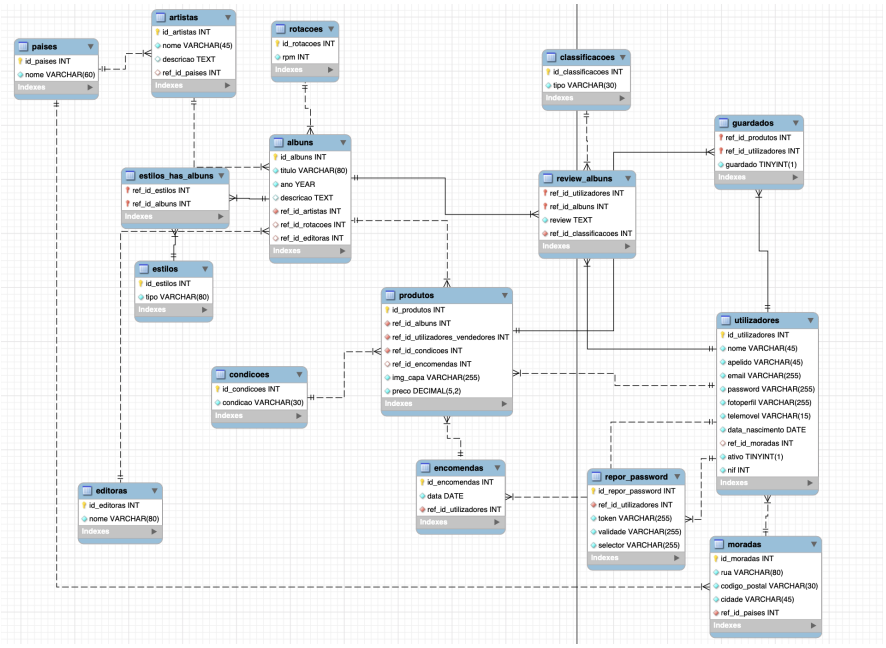
A primeira proposta do modelo, entregue ao professor numa fase inicial, cumpria todos os requisitos da aplicação: comprar vinis; vender vinis; guardar um álbum (adicionar aos favoritos), fazer uma review, ver informação de um álbum (artista, editora, género musical), pesquisar álbuns por estilo, ver histórico de compras, ver os meus anúncios ativos ou vendidos, entre outros. Apesar do receio de não conseguir utilizar na implementação todos os recursos que este modelo me oferecia, decidi avançar, mas sempre disposto a efetuar alterações.



Figuras 1 — Proposta inicial do modelo ER

No entanto, durante todo o percurso não existiram muitas alterações à proposta inicial. Foram apenas duas. A primeira prende-se mais com uma questão da narrativa de uso. Na versão inicial, decidi colocar uma tabela de relação entre os produtos e o utilizadores, onde um *user* podia fazer uma review a um produto, mas só uma por cada. Coloquei ainda a coluna “guardados” de valor booleano, para saber se um *user* tem ou não guardado um álbum. Durante a implementação, percebi que um utilizador não devia fazer review a um produto, mas antes a um álbum. Na verdade, a review deve ser feita à obra musical e não ao objeto físico. Então, mantive esta tabela de relação para os álbuns guardados, e criei uma nova relação de M:N entre álbuns e utilizadores. A segunda alteração ao modelo foi a criação da tabela “repor_password” que, como o nome indica, utilizei para implementar a funcionalidade de reposição de palavra-passe. Todas as tabelas que estavam presentes no primeiro modelo foram implementadas.

A razão pela qual criei uma tabela para os álbuns e outra para os produtos, deveu-se ao facto de não querer criar repetições. Se colocasse todas as informações numa tabela, estaria a repetir informação no caso de um utilizador querer vender um álbum já vendido por outro. Assim, ao criar duas tabelas distintas, não só evito repetições, como posso fornecer ao utilizador informação de álbuns e artistas presentes na base de dados, aumentando a eficiência de criação de um novo produto.



Figuras 2 — Versão final do modelo ER

3. Funcionalidades Implementadas

Penso que cumpri todas as fases de implementação do mini-projeto e foi na quinta fase que investi mais o meu tempo. Esta foi a fase mais desafiante e que me permitiu evoluir. Deste modo, vou focar-me nas funcionalidades mais importantes que implementei nesta fase e que, na minha opinião, tornam este projeto diferenciador.

3.1. Registar, iniciar sessão e recuperação de password

Decidi começar o desenvolvimento do site pela criação de um sistema de login e registo. O registo trouxe alguma complexidade, pois “achei por bem” não obrigar o utilizador a colocar a morada no momento de criação da conta. Isto levou a que tivesse de verificar se os campos de morada estavam preenchidos. No caso de estarem, fazia primeiro INSERT do dados da tabela “morada”, através da função “mysqli_insert_id()” guardava a Chave Primária da instância inserida e, por fim, criava um novo user com o “ref_id_moradas” preenchido. No caso de não estar preenchido, podia fazer logo a criação de um user com o “ref_id_moradas” a NULL.

O grande desafio chegou quando decidi implementar um sistema de recuperação de password. Necessitei de uma API de envio de emails, para enviar um email ao utilizador com um token de recuperação. Utilizei a sugestão do professor: Twilio SendGrid[2]. Criei também uma nova tabela, para me permitir saber de quem era o utilizador que tinha um token atribuído, dentro de uma validade de trinta minutos.

3.2. Vender um álbum

A página de venda é um exemplo de uma funcionalidade que à primeira vista pode parecer de fácil implementação, mas ao entrar a fundo no desenvolvimento apercebemo-nos das dificuldades.

Como existe a possibilidade de dois utilizadores diferentes venderem álbuns de um mesmo intérprete, ou até o mesmo álbum, percebi que fazia sentido dar a possibilidade ao utilizador de escolher artistas já existentes na base de dados. Não só para facilitar a criação de um anúncio, mas também evitar repetições de instâncias. No caso de ser escolhido um artista já existente, existe também a possibilidade de o álbum que o user quer vender já estar presente na base de dados. Ainda existe o caso de o utilizador querer vender uma obra de uma artista “novo”, e sabemos à partida que não temos nenhum álbum do mesmo na BD.

Estas pequenas questões levaram a que a implementação desta ferramenta fosse de uma complexidade superior. A maioria do campos de informação do álbum são opcionais. Por exemplo, um utilizador ao escolher um artista já existente, já não necessita de preencher os campos de criação de artista.

A ferramenta baseou-se ainda em duas outras tecnologias: Javascript e AJAX. Javascript para esconder e mostrar os campos conforme as opções que o user escolhia. Por exemplo, se eu escolher um álbum da BD, não pretendo ver os inputs da criação de um novo álbum. Utilizei também AJAX porque não queria que o utilizador, após escolher o intérprete, visse todos os álbuns da BD, mas apenas os do artista escolhido. Desta forma, consigo que os conteúdos se alterem de forma dinâmica e muito mais intuitiva.

Outro exemplo da complexidade que existe numa questão que parece simples na teoria, foi a funcionalidade de adicionar estilos a um álbum. Um utilizador pode atribuir vários géneros musicais a um álbum, através de um grupo de checkboxs. Então e se sentir que nenhum género se adequa ao seu álbum? Decidi permitir que o utilizador adicionasse um novo género. Então e se o utilizador quiser adicionar vários géneros novos a um álbum? Também decidi permitir esta possibilidade. Como? Através da função explode(), que me permite dividir uma string em partes e transformá-la num array ($outrosgeneros). No input

peço ao utilizador para separar os estilos por vírgulas, e uso-as como “separator”. Após isso, faço INSERT destes novos dados para a tabela “estilos”, onde faço execute da *query* dentro de um foreach do array criado. Ou seja, a repetição da query vai ser executada tendo em conta o número de elementos que o array dos estilos tem. Adicionei novos estilos à tabela, mas ainda não os associei ao álbum criado, na tabela “estilos_has_albuns”. Como posso então saber os id dos estilos que acabei de criar? Dentro do foreach que tinha criado, não só executei a query, mas também criei um array ($array_generos) e faço push de todos os id adicionados, através do mysqli_insert_id(). Por fim, através de outro foreach, mas agora do array “$array_generos”, faço execute da query de INSERT na tabela “estilos_has_albuns”.

```php
if (!empty($_POST["outrogeneroalbum"])) {
    //vamos adicionar um ou mais novos géneros musicais à base de dados...
    $outrosgeneros = explode( separator: ', ', $_POST["outrogeneroalbum"]);

    $query = "INSERT INTO estilos (estilos.tipo) VALUES (?)";
    if (mysqli_stmt_prepare($stmt, $query)) {
        mysqli_stmt_bind_param($stmt, types: 's', &var1: $outrogenero);

        foreach ($outrosgeneros as $outrogenero) {
            if (mysqli_stmt_execute($stmt)) {
                $array_generos[] = mysqli_insert_id($link);
            } else {
                echo "Error:" . mysqli_stmt_error($stmt);
            }
        }
    } else {
        echo "Error:" . mysqli_error($link);
    }

}
```

```php
$query = "INSERT INTO estilos_has_albuns (ref_id_estilos, ref_id_albuns) VALUES (?, ?)";

if (mysqli_stmt_prepare($stmt, $query)) {

    mysqli_stmt_bind_param($stmt, types: 'ii', &var1: $id_estilos, &...: $id_album);

    //vamos relacionar ao album
    if (isset($array_generos)) {
        foreach ($array_generos as $id_estilos) {
            if (!mysqli_stmt_execute($stmt)) {
                echo "Error: " . mysqli_stmt_error($stmt);
            }
        }
    }
```

Figuras 1 e 2 — Código de implementação da capacidade de adicionar vários novos géneros

3.3. Guardados (Favoritos)

Para implementar esta funcionalidade da melhor forma, decidi utilizar AJAX, através do objeto XMLHttpRequest. Assim, posso enviar informação para a base de dados, sem que para isso tenha de saltar para uma página de script e voltar à mesma, onde o utilizador vai notar um refresh da página. Esta escolha permitiu tornar a página mais dinâmica. Comecei por ativar uma função (em javascript) sempre que um utilizador clica numa das *checkboxs* de guardar. Através dessa função enviei argumentos através de dois parâmetros: o primeiro recolhe o estado da checkbox (checked), o segundo recolhe o *value* dessa checkbox, onde está atribuído o id do produto. A função envia a informação através do método send() para a página de PHP “sc_guardar_album”. Nesta página verifico se recebi toda a informação e se existe alguma sessão iniciada. Verifico qual o valor do estado da checkbox e se for “true”, coloco a valor da BD “guardado” a 1, caso contrário coloco a 0. Neste caso não necessito que o script me retorne nenhum valor. Ao clicar numa *checkbox*, ela altera o seu estado “*checked*” (como qualquer checkbox, se estiver *true* passa para *false* ou o inverso). Assim, se voltar a clicar, já envio o outro estado da checkbox. Finalmente, ao fazer refresh da página, no componente do cards dos álbuns, verifico sempre se o álbum está guardado pelo utilizador e coloco “checked” caso se confirme. Não mostro as *checksboxs* de guardar se não existir sessão iniciada.

3.4. Infinite Scroll

A implementação desta tecnologia surgiu de um desafio lançado pelo professor ao ter conhecimento que já me tinha relacionado com AJAX durante o desenvolvimento do projeto. Decorrente da sugestão efetuada, reconheci a importância que esta ferramenta tem

para a eficiência e velocidade de um website. Supondo que temos cem álbuns na nossa BD. Será que faz sentido carregar esses cem álbuns na nossa página? Não. A solução passa então por “chamar” os álbuns em blocos, que só são chamados após o utilizador chegar ao fim da página.

Implementei a ferramenta, utilizando eventos de javascript para ativar uma função que enviava um novo pedido de blocos de quatro álbuns, através de AJAX. Após isso, o PHP vai buscar esse bloco de informação à base de dados, executa o código e devolve o HTML. No entanto, o professor sugeriu que, apesar de ser uma solução, talvez não fosse a mais adequada. Isto porque não tive em conta que um se um utilizador fizesse upload e se existissem utilizadores na página de catálogo nesse momento, se fizessem scroll, existiria uma repetição do álbum. A base de dados já iria ter em conta o novo álbum adicionado e apresentaria o último álbum novamente. O professor sugeriu também fazer a renderização do HTML em Javascript, passando a informação dos álbuns através de JSON, como acontece em frameworks como o React. Através desse objeto de JSON, podia também receber e enviar a informação do último álbum e assim evitar o primeiro problema. Deste modo, o JSON serviu como meio de transmissão entre as duas linguagens. O json_encode() no PHP, criava o objeto e o JSON.parse()[2], permitiu-me “traduzir” para um objeto de javascript. Todas as condições de exibição do card (mostrar ou não o botão guardar, editar ou ver detalhes) passaram então a ser feitas em javascript, dependendo das propriedades booleanas que envio no objeto JSON, como “btnShow” ou “editar”. O código de Javascript para esta funcionalidade está no ficheiro “catalogo.js” da pasta “js”.

3.5. Pesquisa e Filtro utilizando AJAX

A complexidade do projeto aumentou quando decidi criar uma barra de pesquisa em conjunto com um filtro de géneros musicais, que se altera sem atualizarmos a página e funciona e em conjunto com o scroll infinito. Tanto para a pesquisa como para o filtro a abordagem foi idêntica. Recebo a informação (value) da pesquisa e do filtro (os dois porque posso estar a pesquisar um estilo específico ou alterar o filtro de estilos quando tenho um pesquisa ativa) e envio-a através de dois parâmetros para a função “carregaAlbuns()”. Esta função faz o pedido ajax para o ficheiro PHP que acessa a base de dados.

3.6. Pagamento Stripe e TCPDF

Inicialmente descobri a existência de bibliotecas que permitem a criação de pdf’s através de PHP[3]. Pensei que poderia ser interessante implementar na aplicação essa funcionalidade, enviando uma fatura quando um pagamento fosse efetuado. Assim, através do API da Stripe para PHP, decidi implementar a funcionalidade de pagamentos. Envio as informações do álbum e utilizador (email) para API, quando um utilizador clica em “comprar” um álbum. A API processa o pedido e se o pagamento for realizado com sucesso, redireciono para um script onde coloco o álbum como vendido, crio uma fatura em pdf, anexo ao email, envio o email e redireciono novamente, agora para a página de sucesso. Caso o pagamento dê erro, volto à página do álbum que iria ser comprado. A API está no modo “teste” e permite utilizar cartões[4] para testar a funcionalidade. (ver referência 4.)

3.7. Queries

Ao longo do projeto utilizei vários tipos de queries, algumas mais complexas. Segue-se o registo das mesmas.

```sql
SELECT id_estilos, tipo, COUNT(id_estilos)
FROM `estilos`
INNER JOIN estilos_has_albuns
ON id_estilos = estilos_has_albuns.ref_id_estilos
INNER JOIN albuns
ON estilos_has_albuns.ref_id_albuns = id_albuns
INNER JOIN produtos
ON albuns.id_albuns = produtos.ref_id_albuns
WHERE estilos_has_albuns.ref_id_estilos IS NOT NULL AND
produtos.ref_id_encomendas IS NULL
GROUP BY id_estilos
ORDER BY tipo
```

Figura 3 — Query para fornecer as opções do filtro de estilos, na pesquisa

Na página de catálogo, inicialmente estava a utilizar uma query que me dava todos os estilos existentes na base de dados. Percebi que não fazia sentido. Estava a induzir o utilizador em erro, visto estar a dar a opção de escolher estilos sem álbuns. A query tornou-se ainda mais complexa quando quis também informar ao utilizador quantos produtos (e não álbuns) cada estilo tem. Esses produtos não podem ter sido vendidos.

```sql
SELECT AVG(review_albuns.ref_id_classificacoes)
FROM review_albuns
WHERE review_albuns.ref_id_albuns = ?
```

Figura 4 — Query que nos permite saber a média de classificações de um álbum

Utilizei esta query (Fig. 8) para saber qual a média de classificações de um determinado álbum. Após saber esse valor, é exibido o número de estrelas (e meias estrelas) correspondente a essa classificação.

```sql
SELECT artistas.nome, albuns.titulo, produtos.preco, produtos.img_capa, produtos.id_produtos,
guardado, ref_id_utilizadores_vendedores, utilizadores.nome, utilizadores.apelido, fotoperfil
FROM artistas
INNER JOIN albuns
ON artistas.id_artistas = albuns.ref_id_artistas " . $joinestilos . "
INNER JOIN produtos
ON albuns.id_albuns = produtos.ref_id_albuns
LEFT JOIN guardados
ON produtos.id_produtos = guardados.ref_id_produtos AND guardados.ref_id_utilizadores = ?
LEFT JOIN utilizadores
ON utilizadores.id_utilizadores = ref_id_utilizadores_vendedores WHERE " . $pesquisaquery . " "
. $estiloquery . " produtos.ref_id_encomendas IS NULL " . $ultimo_id . " AND ativo = 1
ORDER BY `produtos`.`id_produtos` DESC
LIMIT 0, ?
```

Figura 5 — Query para recolher informação dos álbuns (interface phpMyAdmin para facilitar visualização)

A query que permite exibir o catálogo da loja foi, sem dúvida, a mais desafiante de implementar. Não pela complexidade da mesma, mas pelas variantes que nela existem. São três: o utilizador pode ou não pesquisar, o utilizador pode ou não escolher um estilo e pode ou não ser o primeiro pedido de ajax da página (e não ser enviado o último id do álbum, porque não existe). Estas 2^3 = 8 possibilidades de query criam combinações como: posso pesquisar um álbum, com um estilo selecionado, com o id do último álbum não definido e, quando faço scroll até ao fim da página, passo a ter de definir o último álbum e manter a pesquisa e o estilo ativos. As 8 possibilidades de queries obrigaram também a fazer 8 possibilidades de bind dos parâmetros.

6. Considerações finais

À medida que o projeto se desenrolava, começou a surgir em mim a vontade de vê-lo materializado da forma que idealizei. Não como projeto, pois já tinha os objetivos cumpridos, mas como produto. Penso que a versão final está sólida o suficiente para se tornar numa ferramenta utilizável. Consegui ainda implementar funcionalidades que inicialmente não me tinha proposto e que foram surgindo, muitas vezes pela curiosidade de encontrar soluções mais complexas. Frisar que houve um longo caminho de pequenas adversidades até chegar à versão final. Situações como, por exemplo, não saber que não é possível fazer bind entre '%?%',[5] e repetir o processo de encontrar o erro vezes sem conta até descobrir a solução. Na generalidade, penso que me superei e consegui entregar de forma autónoma um produto com bastante qualidade.

Contudo, ainda existem algumas questões que gostaria de implementar na aplicação. O carrinho de compras e pesquisar por artistas ou por outros filtros são algumas questões que não foram implementadas, devido à ordenação das prioridades que defini e ao curto espaço de tempo disponível. Apesar de serem quase repetições de código já escrito (guardados e carrinho, por exemplo), sinto que poderiam elevar ainda mais o produto.

O balanço final é positivo. Vi neste mini-projeto uma forma de não só mostrar conhecimentos, como também extrair novas aprendizagens e evoluir. Perto do final do projeto, encontrei-me várias vezes na situação de estar a rever código mais antigo e a reescrevê-lo, ao perceber que existia uma solução mais eficaz. Uma prova da evolução que pode advir de projetos como este. Queria também deixar um agradecimento ao professor, pelos desafios que me foi lançando ao longo do projeto. Fui recompensado pelo tempo e empenho investido neste projeto, pois sai dele um produto do qual me orgulho.

7. Referências

1. SendGrid: Email Delivery Service. Consultado a 28 de maio de 2021.
2. JSON.parse(). Consultado a 2 de junho de 2021.
3. TCPDF. Consultado a 27 de maio de 2021.
4. Test your integration. Consultado a 30 de maio de 2021.
5. PDOStatement::bindParam - Manual. Consultado a 1 de junho de 2021.