Programação na Internet

LI52D – Fernando Carvalho

Relatório do Trabalho Prático

Bruno Filipe nº41484

João Gameiro nº41893

Nuno Pinto nº41529

Introdução

Este trabalho consiste numa aplicação web que fornece acesso às funcionalidades de pesquisa de filmes e atores disponibilizadas pelo site TMDb (https://www.themoviedb.org), assim como registo de utilizadores o que possibilita o acesso à criação de listas de filmes e à realização de comentários sobre um filme. As funcionalidades relativas ao utilizador foram implementadas com o auxilio da base de dados não relacional CouchDb (http://couchdb.apache.org/).

O projeto foi organizado em três níveis: *User Interface*; Domínio; Fonte de Dados:

- *User Interface* refere-se à parte visual da aplicação assim como interação do utilizador com a aplicação.

- Domínio refere-se às entidades da nossa aplicação e do serviço que acede à fonte de dados e traduz os dados retornados em entidades.

- Fonte de dados refere-se ao local onde estão armazenados os dados da nossa app. No nosso caso é a API do TMDb e da CouchDb

A ponte entre a parte visual da aplicação e o domínio é definida pelas *routes* da nossa aplicação que correspondem aos *endpoints*.

User Interface

A *user interface* é composta por código *html* (denominado por *view*) e código javascript cliente, responsável pela manipulação visual dinâmica, incluindo pedidos AJAX e alteração de objetos a partir de DOM – *Domain Object Model*.

Para a construção do *html* utilizámos o motor de *templates Handlebars*, importando o módulo hbs para ser adicionado à instância de *express*, tornando-se assim mais um *middleware*. Na resposta do servidor a pedidos que devolvam recursos, as páginas desenvolvidas em .hbs (extensão de ficheiros específicos para o *view engine Handlebars*) são renderizadas juntamente com um objeto que trabalha como contexto, pois no código em hbs são anotadas propriedades que o objeto contexto deve possuir de modo a construir a página específica ao recurso devolvido pelo serviço.

Esta ferramenta fornece a possibilidade de criar blocos auxiliares denominados por *helper*, que permite chamar funções JS a partir de código hbs registados através da função *hbs.registerHelper(<nome do helper>, <função a executar>)*.

Tem como outra possibilidade eliminar código repetido, criando ficheiros chamados de "*partials*" registando estes mesmos ficheiros no momento de adição do *Handlebars* como *view engine* ao *Express* através da função *hbs.registarPartials(<caminho para a pasta onde estão presentes os ficheiros usados como partials>)*.

Os ficheiros relativos às *views* encontram-se na pasta *views*, tendo como subpasta o conjunto de *partials*.

Na pasta *public* encontram-se os ficheiros fornecidos estaticamente, como por exemplo: os ficheiros CSS importados, as imagens utilizadas por *default* (ex: foto de perfil do utilizador, foto genérica de ator e de poster de filme), também o ícone especifico da Aplicação *Web* (*favicon.ico*) e por fim todos os ficheiros de JS cliente utilizados pela interface visual, tanto como resposta ao clique de botões ou no carregamento de determinadas páginas.

Domínio

A camada de domínio é responsável por representar os conceitos abordados na aplicação e traduzir informação vinda da fonte de dados nos modelos definidos.

Modelo

Os objetos de domínio são uma representação do que é retornado da nossa fonte de dados e são populados no *mapper.js.*

Estes objetos são:

- Actor.js;
- CastMember.js;
- Comment.js;
- Director.js;
- Movie.js;
- MovieList.js;
- MovieListItem.js;
- User.js;
- UserList.js;

Serviço

Camada que acede à fonte de dados fazendo a ponte entre o roteamento e os objetos de domínio da aplicação. Recebe pedidos e retorna as entidades populadas de acordo com o formato estabelecido no modelo de domínio.

Na nossa *WebApp* possuímos serviços para gerir os comentários a filmes(*commentService.js*), utilizadores(*UserService.js*), listas de utilizadores(*userListService.js*), *helpers*(*viewService.js*), um serviço com funções que acedem à API(*tmdbService.js*) e outro que contém funções utilitárias(*serviceUtils.js*).

Cache

A nossa *WebApp* possui uma implementação de uma Cache com métodos de acesso(*put*, *get*, *has* e *count*) e cujo objetivo é armazenar os detalhes do filme/ator selecionado pelo utilizador.

Uma cache bem implementada tornará a resposta ao pedido mais rápida, pois quando o utilizador selecionar um filme/ator que já esteja guardado em cache, evita fazer o respetivo pedido à API com que estamos a trabalhar neste projeto.

Memoize

A função *memoize* retorna uma função que pode ser chamada mais tarde, cujo objetivo é:

- Retornar o objeto(no nosso caso, um filme/ator) cujo identificador é recebido como parâmetro, caso este esteja em *Cache*;
- Armazenar o objeto retornado na resposta ao pedido feito à API.

Routing

Routing consiste em traduzir pedidos à aplicação em recursos representados num formato que seja interpretado pelos browsers (neste caso, HTML).

Existem dois tipos de *routing*:

1) *server-side routing* consiste em enviar uma resposta processada pelo servidor ao browser que vai por sua vez ser utilizada como nova página visualizada pelo user;
2) *cliente-side routing* como no caso do server-side recebe um pedido a um recurso e retorna-o no formato pré-estabelecido, mas neste caso não é carregada uma nova página, mas o novo elemento é adicionado à pagina atual.

Na nossa aplicação existem 6 grupos de rotas:

1) índex
   a) `GET /home` – Apresenta a homepage da aplicação
2) actors
   a) `GET /actors/:actorId` – Apresenta um ator com os filmes em que participou
3) Movies
   a) `GET /movies/search` – Apresenta o resultado de uma pesquisa de filmes
   b) `GET /movies/:movieId` – Apresenta os detalhes de um filme, actores, realizadores e comentários ao filme
4) Comments
   a) `GET /comments/:movieId` – Retorna os comentários de um filme
   b) `POST /comments/:movieId` – Introduz um comentário num filme
5) Auth
   a) `GET/POST /auth/signin` – Apresenta a página de *sign in* da aplicação ou faz um pedido de *sign in* à aplicação.
   b) `GET/POST /auth/register` – Apresenta a página de registo ou regista um *user* na aplicação

6) Users
   a) `GET /users/:username – Apresenta pagina de perfil de um user`
   b) `GET /users/:username/lists` – Apresenta listas de um utilizador
   c) `DELETE /users/:username/lists` – Apaga lista
   d) `GET /users/:username/lists/new` – Apresenta pagina com form para criar nova lista
   e) `GET /users/:username/lists/:listId` – Apresenta filmes de uma lista
   f) `POST /users/:username/lists/:listId` – Adiciona filme a lista
   g) `DELETE /users/:username/lists/:listId` – Retira filme de uma lista
   h) `PUT /users/:username/lists/:listId` – Actualiza informaçao da e uma lista
   i) `GET /users/:username/comments` – Apresenta comentarios de um user

Pelo facto de estarmos a utilizar a *framework express*, tivemos oportunidade de criar facilmente um *middleware* com o objetivo de restringir acesso a certas partes da aplicação caso não haja uma sessão iniciada.