UNIVERSIDADE D
COIMBRA

Reactor Core/Web Reactive

Integração de Sistemas

Lino Varela - 2020220433
Nuno Carvalho - 2020219249

Índice

1 Introdução

Este projeto tem como objetivo aprender a desenvolver programas utilizando um modelo de programação reativa, através do desenvolvimento de uma aplicação Web que utiliza a framework *WebFlux* para criar os serviços reativos.

O *Webflux* permite lidar com um grande número de pedidos em simultâneo, sem a necessidade de bloqueios, e utiliza uma estrutura de dados que facilita o processamento assíncrono. No projeto esta framework foi utilizada para a comunicação entre o servidor e o cliente. Ajuda a lidar com pedidos, respostas, acesso aos dados e a manipulá-los de forma mais eficiente.

A aplicação web desenvolvida fornece serviços relacionados com media, utilizadores e a relação entre estes. O servidor apenas utiliza operações CRUD para fornecer os dados.

As informações devem ser guardadas numa base de dados, sendo que o cliente vai buscar a informação necessária a esta através do servidor, e posteriormente adapta a informação recebida para o que pretende analisar. A informação é depois guardada num ficheiro.

2 Desenvolvimento

Para o desenvolvimento do projeto utilizou-se o *Springboot*, que segue uma estrutura simples com uma configuração automática consoante as dependências do projeto, sendo que para gerir as dependências foi utilizado o Maven.

O projeto foi criado utilizando https://start.spring.io/.

2.1 Base de dados

Para guardar todos os dados relativos aos utilizadores e media, utilizou-se a base de dados relacional *PostgresSQl*, sendo primeiro desenvolvido um modelo relacional para a sua criação:

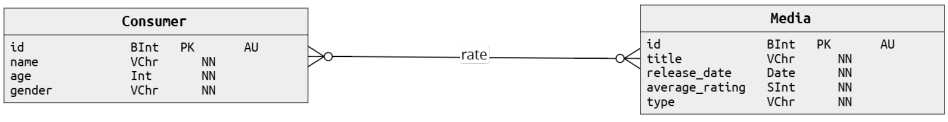

Figura 1: Modelo de entidade conceptual

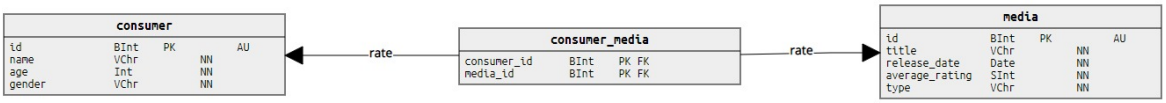

Figura 2: Modelo físico

No sistema, os dados principais são os utilizadores ("consumers") e os media. Os utilizadores podem estar associados a medias e vice-versa. Para gerir e guardar estas relações foi necessária uma tabela. A comunicação entre o servidor e a base de dados é realizada utilizando a biblioteca *R2DBC* (*Reactive Relational Database Connectivity*), que permite executar operações de forma assíncrona. A sua implementação é feita com os repositórios *ReactiveCrudRepository*, que permitem operações CRUD de forma reativa.

A configuração do R2DBC e da base de dados é feita no ficheiro *application.properties*, em que se especifica o URL e informações da DB.

2.2 Servidor

O servidor foi implementado utilizando o *Spring WebFlux*, a framework previamente mencionada, que permite a construção de APIs reativas e escaláveis, garantindo que se possa lidar com múltiplos pedidos ao mesmo tempo de forma eficiente, mesmo com fluxos de dados assíncronos.

No servidor faz-se a gestão das informações sobre media e utilizadores através de controladores, que mapeiam os endpoints da API aos métodos que tratam das operações.

Endpoint	**Operação**	**Descrição**
`/api/media`	GET	Retorna todos os dados media
`/api/media/{id}`	GET	Retorna uma media em específico, utilizando o ID
`/api/media`	POST	Cria um nova media.
`/api/media/{id}`	PUT	Atualiza media através do ID
`/api/media/{id}`	DELETE	Remove uma media (se não existir relações com utilizador)
`/api/consumers`	GET	Retorna todos os utilizadores
`/api/consumers/{id}`	GET	Retorna um utilizador com um ID em específico
`/api/consumers`	POST	Cria um novo utilizador
`/api/consumers/{id}`	PUT	Atualiza um utilizador existente através do ID
`/api/consumers/{id}`	DELETE	Remove um utilizador
`/api/relationships`	GET	Retorna todas as relações entre utilizadores e as media
`/api/relationships/{consumerId}/{mediaId}`	GET	Retorna uma relação em específico entre um utilizador e media
`/api/relationships`	POST	Cria uma nova relação entre um utilizador e uma media
`/api/relationships/{consumerId}/{mediaId}`	DELETE	Remove uma relação entre um utilizador e media

Tabela 1: Lista de Endpoints

Existe uma camada de serviços que trata da lógica da aplicação, ou seja, manipula os dados e realiza as operações específicas, comunicando com os repositórios para o fazer.

Foi implementado logging para mostrar as opções realizadas, potenciais erros e falhas,

utilizando-se o *Lombok* e *SLF4J*.

2.3 Cliente

A implementação do cliente é a parte mais relevante do projeto, pois é aqui que se vai buscar dados ao servidor e, através de diversas queries, manipular os dados de forma a obter o pretendido.

O cliente utiliza o *WebClient* do Spring que simplifica a comunicação com o servidor e permite operações assíncronas e não bloqueantes, melhorando a eficiência da aplicação.

Em primeiro lugar, foi realizada a configuração inicial do *WebClient* e, posteriormente, efetuaram-se as operações.

Todas as operações criadas utilizam o método *webClient.get()* para aceder aos endpoints do servidor e obter as informações necessárias. A resposta do servidor é então manipulada de maneira eficiente, utilizando programação reativa, para obter o resultado pretendido. Os resultados são sempre escritos em ficheiro de texto, depois de concluídas as operações.

O método *bodyToFlux* é utilizado em todas as funções para converter a resposta do servidor para o objeto correspondente (fluxo reativo). Além disso, nas operações é utilizado o *CountDownLatch*, para garantir que todas são executadas, evitando bloqueios. Esta funcionalidade também garante que todas operações são concluídas.

De seguida, são explicados os pontos importantes do desenvolvimento das queries, com maior detalhe nas mais complexas. Como mencionado anteriormente, todas as queries utilizam inicialmente o *bodyToFlux* para obter os objetos do servidor.

A primeira query utiliza o método *map* para obter a informação da base de dados sem realizar qualquer tipo de alteração. Na segunda é aplicado o método *count* para contar o número total dos media.

Na terceira query, é aplicado um filtro para garantir que apenas os objetos com um *averageRating* superior a 8 são obtidos. Para tal, utiliza-se o método filter().

Na query seguinte, foi necessário fazer uma filtragem de duplicados, usando o método *distinct*, que garante que os valores retornados são únicos. Para tal, é apenas mapeado o ID da media.

Na quinta query aplica-se um filtro mais elaborado em que se verifica se a data de lançamento do media foi nos anos 80. Posteriormente, faz-se o ordenamento do output por ordem crescente de datas com o método *sort*.

A sexta query utiliza o método *reduce*, que permite transformar o fluxo de dados numa *Arraylist* do tipo *Double*. Este passo é necessário para obter os ratings de cada media e adicioná-los à lista. De seguida, calcula-se a média somando os ratings e dividindo pelo número total de elementos e o desvio padrão com base na variância, que tem em conta a diferença entre cada rating e a média.

A sétima query obtém o objeto media mais antigo. Aqui apenas é realizada uma operação *sort* para ordenar as datas por ordem de lançamento. Depois de ordenada a lista, utiliza-se a operação *next()* para se obter o primeiro objeto, que corresponde à media mais antiga.

Na oitava query pretende-se calcular o número de utilizadores por cada objeto media. Após obter todos os dados relativos à media, utilizamos o *flatmap* para que, para cada

media, se consiga obter as relações a elas associadas de forma assíncrona. São filtradas as relações para garantir que os utilizadores estão realmente associados à media em especifico. O *count* é utilizado para fazer a contagem, que depois é mapeada junto ao ID da media. Posteriormente, é utilizada a operação *reduce* que junta os resultados da soma do número total de utilizadores de todas as medias, sendo o número de medias também contado. Após este processo, calcula-se a média de utilizadores por media.

A nona query visa obter dados dos utilizadores de cada media, ordenados por idade. Ao obter os dados da media, utilizamos o *flatmap* para buscar as relações e filtramos para garantir que apenas são tratadas as relações que envolvem a media que se está a processar. De seguida, outro *flatmap* é utilizado para obter as informações de cada utilizador (através do ID). Obtém-se com estas operações um fluxo de dados com os consumidores relacionados com a media. Seguidamente, agrupam-se (*groupBy*) os resultados pelo ID da media e em cada grupo utiliza-se o sort para fazer a ordenação por idade. Por fim, faz-se mapeamento dos dados para se obter o resultado.

A última query tem como objetivo obter uma lista com todos os utilizadores e as medias a que estes estão associados. Após obter os dados dos utilizadores, utiliza-se um *flatmap* para obter as relações. Filtra-se e, novamente, faz-se uso de outro *flatmap* para buscar as medias associadas a cada relacionamento. Assim, consegue-se obter a media a que um user está subscrito.

O último método a mencionar é o *fetchMediaWithRetries*, que representa a primeira query desenvolvida e serve para fazer uma simulação da conexão a falhar e consequentes tentativas do cliente de se reconectar.

3 Conclusão

A implementação da aplicação utilizando *Spring WebFlux* e programação reativa permitiu construir um sistema eficiente para consultar e manipular dados provenientes de uma API REST. Através de operações assíncronas e não bloqueantes, como o *map*, *flatMap* e *reduce*, foi possível realizar tarefas complexas, como o cálculo de média e desvio padrão dos ratings, a obtenção do item de media mais antigo e o cálculo do número médio de utilizadores por item de media.