UNIVERSIDADE D
COIMBRA

Web Reactive

Integração de Sistemas

João Laranjeiro, Marcelo Gomes

Emails: {joaolaranjeiro, marcelogomes}@student.dei.uc.pt

1. Introdução

Este projeto tem como objetivo desenvolver uma aplicação reativa com Spring WebFlux, que expõe serviços para gerir conteúdos de *media* e interações de utilizadores.

Neste relatório, abordaremos a implementação da aplicação, destacando a estrutura da arquitetura e as funcionalidades desenvolvidas. Também discutiremos os desafios enfrentados durante o desenvolvimento e as soluções encontradas, bem como as lições aprendidas ao longo do processo. O objetivo final é fornecer uma visão clara da aplicação desenvolvida, evidenciando a eficácia da programação reativa em cenários de gestão de dados em tempo real.

2. Servidor

Neste capítulo, será apresentada a implementação do servidor, descrevendo de forma resumida a sua estrutura, abordando os mecanismos de logging e o tratamento de dados utilizados.

2.1. Estrutura do Servidor

A estrutura do servidor foi organizada por uma série de pastas e ficheiros organizados para lidar com as diferentes responsabilidades (seguindo a arquitetura em camadas), incluindo controlo de rotas HTTP, manipulação de dados, tratamento de exceções, entre outros.

- **Api:** Contém os *controllers*, responsáveis por definir as rotas HTTP e direcionar os *requests* para as camadas de serviço apropriadas: ***RelationshipController***, ***MediaController***, ***UserController***

- **Service:** Contém as classes de serviço, onde está implementada a lógica de processamento de dados. Cada serviço é responsável por uma entidade específica e executa operações mais complexas que envolvem a lógica de aplicação.

- **Persistence:** A camada *persistence* é responsável pela interação com a base de dados, utilizando Spring Data R2DBC para acesso reativo.
    - **Entity:** As *entities* representam as tabelas da base de dados e são mapeadas para classes Java.
    - **Repository:** Os repositórios fornecem métodos para operações CRUD reativas.

2.2. Estrutura das Tabelas e Relações

Para armazenar e gerir as entidades de *medias* e utilizadores, a aplicação utiliza três tabelas:

- **Tabela de *Media (Media)*:** Armazena dados sobre cada item de *medias* (filme ou série) com os seguintes campos, *Identifier, Title, Release Date, Average Rating* e *Type.*
- **Tabela de Utilizadores (*Users*):** Contém dados sobre cada utilizador da aplicação com os campos, *Identifier, Name, Age e Gender.*
- **Tabela de Relacionamento (*MediaUsers*)**: Esta tabela funciona como uma tabela de ligação que suporta o relacionamento entre as outras duas tabelas, permite que um item de *media* esteja associado a vários utilizadores e que um utilizador tenha múltiplas *medias* associadas. Inclui dois campos: *mediaIdentifier* e *userIdentifier*.

2.3. Endpoints e Operações CRUD

Os endpoints da aplicação foram definidos para garantir o acesso eficiente aos dados e a manipulação das entidades.Todos os endpoints seguem o padrão de operações CRUD.

As operações CRUD para ***User*** e ***Media*** funcionam de forma similar, permitindo criar, ler, atualizar e excluir dados dos utilizadores. O ***CreateUser/Media*** insere um novo item na base de dados, enquanto o ***ReadAllUser/Media*** lista todos os utilizadores e *medias*, e o ***ReadSpecificUser/Media*** permite consultar um utilizador ou *media* específica. A operação ***UpdateSpecificUser/Media*** altera os dados do item existente, e por fim, o ***DeleteSpecificUser/Media*** exclui o item desejado, desde que este não esteja associado a nenhuma *media* ou utilizador.

Quanto à tabela ***MediaUser***, que gere a relação *many-to-many* entre ***User*** e ***Media***, o ***CreateRelationship*** cria uma nova associação entre um utilizador e uma *media*. O ***ReadRelationship*** retorna os identificadores das entidades associadas, sem detalhar os dados completos, e o ***DeleteRelationship*** remove a associação entre os mesmos.

Esses endpoints e as suas respetivas operações CRUD garantem que o servidor forneça todas as funcionalidades básicas necessárias para a gestão das entidades.

2.4. Logging e tratamento de dados

Utilizamos o SLF4J para implementar o sistema de logging da aplicação. O SLF4J é utilizado para registar mensagens de *log* em diferentes níveis (INFO, DEBUG, WARN, ERROR), e cada classe de serviço e controlador possui uma instância dedicada do *logger* para registar eventos importantes. Assim, garantimos o registo consistente de informações, erros e exceções, facilitando a monitorização e depuração da aplicação.

3. Cliente

Este capítulo aborda a implementação do cliente, que realiza consultas ao servidor para acessar e manipular dados. O cliente utiliza *WebClient* para realizar chamadas reativas aos endpoints definidos e, para cada operação, foram aplicadas estratégias de otimização com o objetivo de melhorar a performance.

3.1. Configuração Inicial e Estrutura Geral

O cliente implementado utiliza o *WebClient* para realizar chamadas HTTP reativas aos endpoints do servidor, com operações organizadas em métodos específicos na classe ***ReactiveService***. As funções ***fetchAllMedia***, ***fetchUsersByMedia*** e ***fetchAllUsers*** são responsáveis por obter os dados do servidor, utilizando o *WebClient* com suporte a *retrys* (***retryWhen***) e tratamento de erros (***doOnError***). Métodos como ***getMedia80s*** e ***ratingAvgStdMedia*** processam os dados recebidos, aplicam a lógica necessária e escrevem os resultados em ficheiros utilizando a classe ***FileWriter***, um método assíncrono de escrita de ficheiros que encontrámos no Stack Overflow. A sincronização das operações é feita pela classe ***Client***.

3.2. Implementação dos requisitos

A seguir, são descritos os requisitos funcionais implementados pelo cliente para realizar operações específicas de consulta e manipulação de dados relacionados a *medias*. Cada subsecção apresenta o objetivo da consulta, a lógica de implementação e as técnicas de otimização aplicadas.

3.2.1. Consulta de Títulos e Datas de Lançamento das *médias*

A implementação deste requisito inicia-se com a criação de um fluxo reativo, que é gerado a partir da chamada ao *endpoint* correspondente no servidor. Esse *endpoint* retorna uma lista de objetos contendo os títulos e as datas de lançamento das *medias*. Em seguida, para otimizar o processamento, utiliza-se o operador *map* com o objetivo de extrair apenas as informações necessárias, como o título e a data de lançamento. Após essa transformação, o resultado é guardado num ficheiro.

3.2.2. Contagem total de *medias*

No segundo requisito, o método começa por chamar a função ***fetchAllMedia***, que retorna um *Flux* de todas as *medias*. Em seguida, aplica o operador *count*, que transforma o fluxo num valor único **(Mono<Long>)** com a quantidade total de *medias*. Por fim, esse número é formatado numa *String* e gravado no ficheiro pretendido utilizando o método ***fw.writeRows***.

3.2.3. Contagem total de *medias* com um *rating* médio maior que 8

O método chama a função ***fetchAllMedia*** como o requisito anterior, de seguida utiliza o operador ***filter*** para manter apenas as *medias* cujo rating médio seja maior que 8. Depois de filtrado, o operador *count* é aplicado com o objetivo de obter a contagem das *medias* com boa avaliação. Por fim, esse número também vai ser formatado numa *String* e gravado no ficheiro pretendido.

3.2.4. Contagem total de *medias* que estão subscritas

Novamente o método chama a função ***fetchAllMedia***, para cada *media*, o método utiliza o operador *flatmap* para chamar a função ***fetchUsersByMedia*** , que retorna um fluxo dos IDs de utilizadores associados a essa *media* específica. De seguida, faz-se contagem total de utilizadores associados a cada *media* utilizando o operador ***count*** juntamente com o ***filter(userCount -> userCount != 0)***, de modo a considerar apenas as *medias* que possuem pelo menos um utilizador associado.
Por fim, depois de processadas todas as *medias*, aplica-se o operador ***count*** novamente para obter o total de *medias* com subscritas e escrever esse valor no ficheiro.

3.2.5. Dados das *medias* dos anos 80

No quinto requisito, depois de obter todas as *medias* disponíveis é aplicado um filtro para selecionar apenas as *medias* com datas de lançamento entre 1980 e 1989. Após a filtragem o método ***sort*** organiza os itens resultantes em ordem decrescente de avaliação média, utilizando o comparador ***Double.compare(m2.getAverage_rating(), m1.getAverage_rating()).*** De seguida, o operador ***map*** formata uma frase para conter todos os dados das *medias* para no fim serem escritos no ficheiro.

3.2.6. Média e desvio padrão do rating de todas as *medias*

Este método começa também por obter o fluxo de todas as *medias* e, em seguida, aplicar o ***map*** para extrair o rating médio ***getAverage_rating*** de cada *media*. Os rating são acumulados através de um ***reduce***, que usa um vetor ***float[]*** para armazenar a soma das avaliações, a soma dos quadrados das avaliações, e a contagem total de avaliações.

Logo de seguida, um operador ***map*** é aplicado para calcular a média e o desvio padrão. A média é obtida dividindo-se a soma dos ratings **(acc[0]) / (acc[2])**. A variância é calculada pela fórmula **(acc[1] / acc[2]) - (mean * mean)**, e o desvio padrão é obtido tirando a raiz quadrada da variância.

3.2.7. O nome da *media* mais antiga

Depois de obtidas todas as medias, utiliza-se o operador ***reduce*** para comparar as datas de lançamento de cada item e identificar a *media* mais antiga. Este operador utiliza dois elementos **m1** e **m2** e, com o método ***isBefore***, compara as datas de lançamento e retorna o item mais antigo entre os dois.

Depois de identificada a *media* mais antiga, o operador ***map*** formata uma frase com os dados da mesma e no fim esta é escrita no ficheiro.

3.2.8. O número médio de utilizadores por *media items*

Para cada item de *media*, é usado um ***flatMap*** para ter acesso ao método ***fetchUsersByMedia***, que retorna a quantidade de utilizadores associados a essa *media* específica. De seguida, cada contagem de utilizadores é convertida para um *array* de dois valores: a contagem do item e um contador do próprio item **[userCount, 1]**.
O próximo passo é somar os valores da contagem de utilizadores e o total de itens de *media* utilizando o ***reduce***. O array acumulador (acc) armazena o total de utilizadores em acc[0] e o total de itens de *media* em acc[1]. Por fim, é calculada a média de utilizadores por item de *media*, que é obtida pela divisão do total de utilizadores pelo total de *medias* e escrever esse valor no ficheiro correspondente.

3.2.9. Nome e número de utilizadores por *media,* ordenados pela idade do utilizadores de forma descendente

Depois de obtidas todas as *medias*, o método ***flatMap*** é utilizado para obter os identificadores dos utilizadores associados à *media* através do ***fetchUsersByMedia***. De seguida, para cada identificador de utilizador, são obtidas as informações completas do mesmo através do método ***fetchAllUsers*** e utilizando o operador ***filter*** ficar apenas com o utilizador correspondente ao identificador.
Estes utilizadores são ordenados por idade em ordem decrescente usando o método ***sort***.
Depois de ordenados, a lista de utilizadores é convertida numa string formatada utilizando o operador ***reduce.*** Por fim, cada item é mapeado e formatado com o título e a lista de utilizadores associados. Caso não haja utilizadores, o campo Users exibirá "None".

3.2.10. Dados completos de todos os utilizadores, incluindo os nomes das *medias* subscritas

Neste último requisito, a função ***fetchAllUsers*** obtém todos os utilizadores e, para cada utilizador, utiliza a operação ***flatMap*** para obter os itens de *media* associados, através do método ***fetchMediaByUser***. De seguida, é aplicado outro *flatMap* para obter os detalhes dos itens de *media*, utilizando o método ***fetchAllMedia***. Os itens de *media* são então filtrados pelo id correspondente usando a operação ***filter***. Os títulos dos itens de *media* são agregados numa única string com a operação ***reduce***, que concatena os títulos separados por vírgulas. Por fim, a operação map é utilizada para formatar os dados completos do utilizador, incluindo os títulos das *medias* subscritas, numa string detalhada.

3. Conclusão

O desenvolvimento desta aplicação reativa com Spring WebFlux proporcionou uma visão ampla sobre a programação reativa e as suas vantagens na integração e gestão de dados em tempo real. Através de uma arquitetura organizada, conseguimos implementar um sistema eficiente para gerir conteúdo de *medias* e interações de utilizadores.

Ao longo do projeto, enfrentamos desafios e obstáculos que nos levaram a desenvolver as nossas habilidades de programação e resolução de problemas. No entanto, com trabalho em equipa e dedicação, conseguimos superar esses obstáculos e alcançar um resultado final satisfatório.

Em última análise, este projeto não apenas permitiu aplicar os conceitos aprendidos na aula, mas também nos proporcionou uma oportunidade valiosa para explorar e entender melhor o funcionamento de aplicações reativas. Estamos satisfeitos com o resultado alcançado.