Descrição do algoritmo (Java)

Para desenvolver o algoritmo pedido - devolver uma paginação em números e em String (quando aplicável) - decidi optar por implementar três classes: ***Pagination***, ***Page***, ***RestPaginationController.*** Sendo que na classe Pagination está implementado o método responsável por criar a lista que terá os elementos que vão ser apresentados consoante os valores atribuídos a ***current_page***, ***total_pages***, ***boundaries*** e ***around***. Optei por criar uma REST API, uma vez que o enunciado do exercício refere que este algoritmo seria utilizado numa ***Web Application***.

Class Pagination

Nesta classe defini apenas uma propriedade (***ELLIPSIS)***, e uma vez que o seu valor nunca se ia alterar, defini a mesma como constante e inicializei com o valor de "…". De seguida, comecei por implementar o método createFooter que será responsável por criar a lista de páginas consoante os valores atribuídos às variáveis a cima referidas.

Método createFooter:

Este método recebe quatro parâmetros (current_page, total_pages, boundaries e around) em que todos são do tipo ***int*** e retorna uma ***Linkedlist*** parametrizada com ***String***. Os valores passados como argumentos a este método servem de base para a avaliação de quais os valores que são apresentados como limites(início e fim) assim como as páginas à esquerda e direita da current_page - sendo que todas as demais deveriam ser representadas como uma única ocorrência de "…". Assim, para definir um valor que me permitisse manipular as páginas que estariam ou não à esquerda e direita de current_page, guardei numa variável(***pagesAtLeft***) do tipo int, o cálculo de current_page - around(para as páginas à esquerda) e noutra variável(***pagesAtRight***) current_page + around(para as páginas à direita). Seguindo o mesmo raciocínio, decidi guardar numa variável(***pagesAtEnd***) o cálculo de total_pages - boundaries. Para as páginas iniciais(***pagesAtBeginning***), utilizei a propriedade boundaries em si, uma vez que defini que nunca seria menor que 1, sendo que a nível de paginação faria sentido aparecer sempre, pelo menos, a primeira página. No entanto, antes destes cálculos e para garantir que o utilizador desta REST API não colocasse números inferiores a 1, criei uma condição que preveja essa situação e atire uma ***IllegalArgumentException***.

Tendo em conta que teria que apresentar uma sequência de números e/ou ***String***(caso existisse essa necessidade) optei por utilizar uma Linkedlist - cujo o nome é ***listOfPages*** - como estrutura de dados para poder guardar os elementos que serão apresentados e manipulá-los com relativa facilidade.

Para adicionar à listOfPages apenas e só os valores que satisfaçam as condições requeridas no enunciado, utilizei um ***for loop*** que percorre a totalidade das páginas(***total_pages***) partindo de 1 - isto deve-se ao facto de querer que o primeiro elemento da lista seja sempre 1 - no qual estipulo três condições que servirão o propósito de definir os critérios para introduzir elementos na lista:

1) se o valor de ***i*** é igual ou menor que o valor atribuído a boundaries, este deve ser adicionado à ***listOfPages*** - por exemplo, se boundaries for igual a 2, o valor de 1 e 2 será adicionado à lista;

2) se o valor de ***i*** for maior que o valor em pagesAtEnd, este também será adicionado à lista - mantendo o exemplo do valor de boundaries igual a 2, pagesAtEnd teria o valor de 8, garantindo a presença dos dois últimos valores 9 e 10, caso total_pages for igual a 10;

3) se o valor de ***i*** for maior ou igual a pagesAtLeft **E** menor igual a pagesAtRight garanto que os valores posicionados à esquerda e à direita de current_page são adicionais à lista, assim como o próprio valor de current_page.

No entanto, como o método espera retornar uma ***Linkedlist*** parametrizada com ***String***, teria que converter os valores adicionados do tipo ***int*** em ***String***, daí utilizei o método ***Integer.toString***.

Após este processo, resta colocar o valor de ELLPISIS invés dos valores que não preenchem as condições - por exemplo, {1, 2, …, 5, 6, 7, …, 9, 10}. Para resolver esta situação utilizei o ***else if***, para confirmar que o último elemento colocado na lista era igual a "…", caso fosse utilizei a ***keyword continue*** de forma a ignorar o passo de colocar novamente o valor "…" e assim prosseguir com o loop. Caso nenhuma das condições em cima fosse verificada, utilizei o ***else***, para então adicionar à lista o valor de ELLIPSIS - desta forma preveni este tipo de erro: {1, 2, …, …, 5, 6, 7, …, 9, 10}.

Por fim, o método retorna uma ***Linkedlist***, sendo este método invocado na classe ***RestPaginationController***, onde contém um método(listPages) que lidará com um ***request*** do utilizador, fornecendo uma ***response*** ao mesmo.

Relativamente a ***Unit Testing*** utilizei o método ***assertEquals*** fornecido pela framework ***JUnit*** na classe ***PaginationTest*** de forma a comparar o resultado esperado com o output que o método createfooter

daria em diferentes situações - por exemplo valores em que o around é superior ao total_pages ou se o current_page for menor que 1.

Descrição do algoritmo (GOlang)

Uma vez que idealmente seria entregar a resolução deste exercício em GOlang, e tendo em conta que é uma linguagem que estou a começar a aprender, decidi tentar implementar o que tinha feito no projecto Java num projecto GOlang. Optei por implementar 4 ficheiros: main.go, pagination.go, restpaginationcontroler.go.

Pagination.go

Neste ficheiro está definido a lógica relativamente ao algoritmo, que a esse nível está semelhante ao projecto em Java, com as diferenças que uma linguagem diferente acarreta. Neste caso, como era minha intenção devolver uma ***string slice*** na resposta ao request do utilizador, expondo-a através de uma REST API, decidi criar uma ***struct*** do tipo ***Footer*** que contivesse a referida lista como a sua única propriedade. Esta intenção justifica a inclusão da anotação ` ***json:"list"*** ` e materializa-se na implementação do RestController que retorna precisamente uma codificação/serialização em json de um objeto deste tipo Footer. Na função ***newFooter***, a lógica é praticamente igual, apenas com a diferença da utilização de métodos que executassem a mesma lógica - por exemplo a utilização do método ***append***(adicionar à lista), do ***strconv.Itoa***(converter int em string) e do cálculo do último elemento adicionado à lista ***(list[len(list)-1]***. Por fim o método retorna então um objecto do tipo ***Footer***, que vai conter a list com os elementos adicionados e é invocado no ficheiro ***restpaginationcontroller.go*** na função ***getPages***.

Relativamente a unit testing, segui o mesmo raciocínio anteriormente explicado e implementado no mesmo exercício em Java/jUnit, porém utilizando a biblioteca que Golang disponibiliza à partida para fazer todas as verificações.