Trabalho Final de Computação na Nuvem

Joana Campos, A44792

Nuno Cardeal, A44863

Carolina Couto, A44871

Professor: José Simão

20 de junho 2021

Índice

1. Introdução

Este trabalho tem como objetivo realizar um sistema que dado um determinado ficheiro de imagem detetando as suas características, em inglês, traduzindo-as depois para português e guardando-as num documento da base de dados *Firestore* da *Google Cloud Platform*. Devendo também ser possível filtrar os ficheiros de imagem submetidos por característica, dado um determinado intervalo de datas.

2. Contrato

O primeiro ponto a ser implementado é o contrato que contém um ficheiro proto com um serviço e cinco mensagens. No serviço estarão presentes três funções. A primeira representa a função de submissão de uma imagem que é um caso 3, ou seja, são feitos vários pedidos pelo cliente obtendo apenas uma resposta final do servidor. A segunda representa a função de obtenção das características e das respetivas traduções que é um caso 1, ou seja, é feito um pedido no cliente e é obtida uma resposta do servidor. A terceira representa a função de filtragem de ficheiros que é também um caso 1.

A primeira mensagem a ser implementada é a *ImageRequest* que contém um *array* de *bytes* para enviar os blocos de *bytes* da imagem a ser submetida, uma *string* com o nome do *blob* para onde será submetida a imagem e uma *string* com o tipo do conteúdo do ficheiro de imagem a ser submetido. A segunda mensagem é *ImageResult* que contem uma *string* com o *id* da imagem submetida. A terceira mensagem é *Labels* que contem um *array* de *string* com as caraterísticas e um *array* de *string* com as características traduzidas. A quarta mensagem é *FilterRequest* que contem uma *string* com a data inicial, uma *string* com a data final e uma *string* com a característica a ser filtrada. A quinta e última mensagem é *FilterResult* que contem um *array* de *string* com os nomes dos ficheiros que satisfazem o filtro.

´

3. Servidor

Com o contrato definido, de seguida irá ser implementado o servidor que irá implementar as funcionalidades do sistema de forma a poder ser utilizado pelas aplicações cliente.

A aplicação servidora irá conter ficheiros java com os serviços necessários para as funcionalidades, neste caso será necessário ter acesso ao *Cloud Storage,* ao *Cloud Pub/Sub* e ao *Firestore.* Cada uma destas classes irá incluir um método estático *init* que irá estar encarregue de inicializar o serviço em questão por forma a poder ser utilizado pela aplicação.

No *main* da aplicação será inicializado o servidor, assim como será chamado o método *init* de cada serviço, descrito anteriormente, de forma a poder ser utilizado. No final é chamado o método *awaitTermination* para poder esperar que todas as tarefas terminem antes de fechar a ligação.

De seguida é feito *override* dos métodos definidos no contrato construído.

3.1 Submissão de um Ficheiro Imagem

O método definido *uploadImage* estará encarregue de guardar um ficheiro imagem no *Cloud Storage.* Este irá receber o ficheiro em *stream* de blocos e para tal será necessário criar um *observer* que irá implementar *StreamObserver* que estará encarregue de colocar os diferentes *stream* de blocos num só *blob* do *Cloud Storage.* No método *onNext* que irá receber o *stream* será chamado o método *uploadToStorage* implementado na classe *CloudStorageService.* Este método irá utilizar um *WriteChannel* para poder escrever num *blob* e que caso esse *channel* esteja vazio então será criado um *blob,* com o nome do ficheiro de imagem que será publicado, no *bucket* previamente criado na *Cloud*. De seguida o *WriteChannel* irá conter o *writer* que criará o *blob* e retornará um canal para poder escrever o seu conteúdo. Caso o método *uploadToStorage* seja chamado e já exista um canal para escrever então apenas será adicionado o novo conteúdo ao existente. Esta forma foi utilizada visto que o servidor irá receber o conteúdo do ficheiro em vários blocos e assim poderá colocar no *blob* à medida que os recebe. De seguida foi definido o método onError que irá colocar no *observer* que receber para colocar a resposta de volta para o cliente a causa do erro.

Por fim, no método *onCompleted* é fechado o *WriteChannel* que foi criado e é definida a resposta para enviar de volta para a aplicação cliente. Neste caso é construído um *ImageResult* que irá conter o id do pedido que será uma conjunção do nome do *bucket* e o nome do *blob* sendo os dois separados por um hífen. Com o resultado contruído é só colocar no *observer* recebido para enviar a resposta de volta para o cliente chamando o método *onNext* desse mesmo observer com o resultado contruído e o método *onCompleted.* Depois do ficheiro ser guardado no *Cloud Storage* será enviado para um tópico *Pub/Sub* criado previamente com o nome de *topicworkers* com o identificador do pedido, nome do *bucket* e do *blob* para processamento de obtenção de *labels.* Para tal foi criado o método *publishMessage* na classe que representa o serviço do *Pub/Sub*. O método irá buscar o tópico *topicworkers* e de seguida criar um *publisher* de forma a poder publicar a mensagem. Para criar a mensagem será criada uma variável *PubSubMessage*

que será contruída com uma *string* que irá conter a informação mencionada anteriormente, assim como os atributos que irão conter o nome do *blob* juntamente com a *string "*-metadata*"*. Finalmente é criado um *ApiFuture* do tipo *string* que irá chamar o método *publish* do *publisher* criado anteriormente que irá receber o *builder PubsubMessage* contruído. Isto tudo encontra-se dentro de um *try* e no *finally,* caso o *publisher* não se encontre nulo é feito o fecho e ficará à espera que as tarefas terminem chamando o método *awaitTermination.*

3.2 Listagem das Características e Traduções numa Imagem

Para a listagem de características utilizou-se o método *getLabelsList* que irá receber um *ImageResult* como *request* que irá conter o identificador do pedido para identificar de qual imagem se deve apresentar as características e respetivas traduções. Para tal é utilizado o método *addLabelsAndTranslations* da classe criada *FirestoreService* para representar o serviço do *Firestore* onde, com o *id* recebido no pedido irá-se buscar o documento que contém esse *id* e, caso um documento com esse documento exista então irá ser guardado em duas listas do tipo *string*, *originals* para guardar as características em inglês que se encontra no campo *labels* do *Firestore* e *translated* para guardar as traduções dessas mesmas características que se encontram com campo *translated* do *Firestore.* Caso o documento não exista irá ser lançada uma exceção criada especificamente para este problema chamada *DocumentNotFounException* que irá estender de uma *BaseException* também criada que irá receber uma mensagem em forma de *string* e retornar essa mesma mensagem com o método *getMessage.* Feito o método será agora colocado no *observer* recebido a resposta em que se irá passar um *builder* do tipo *Labels* definido no contrato em que se adiciona as características e traduções colocadas nas listas *originals* e *translated,* respetivamente. No fim, será colocado esse *builder* no método *onNext* do *observer* e de seguida chamada o método *onCompleted* do mesmo.

3.3 Filtração de Ficheiros Armazenados no Sistema

Na filtração do nome de ficheiros mais uma vez acedeu-se ao serviço do *Firestore* com um método criado *getFilteredFilesName* em que irá receber as duas datas e a característica. Nesse método é criado uma query que irá buscar a coleção criada previamente no serviço *Firestore* do *Google Cloud* com o nome *"image-labels"* e de seguida é utilizado o método *whereGreaterThanOrEqualTo* em que é verificado se a data de algum documento presente na coleção no campo *uploadDate* é maior que a data inicial passado no pedido, o oposto é feito com a data final em que é utilizado o método *whereLessThanOrEqualTo* para verificar se algumas das datas presentes nos documentos da coleção que retornaram à chamada do método anterior (*whereGreaterThanOrEqualTo)* no campo *uploadDate é* menor que a passada. Finalmente ainda é verificado se alguns dos documentos retornados da condição anterior contem na lista *tranlated,* que contém as características traduzidas, a características passada no pedido por uma aplicação cliente. Com a *query* feita é feito um *query.get()* que retornará o resultado da *query* e de seguida irá-se percorrer os documentos que essa

mesma *query* contém chamando o método *querySnapshot.get().getDocuments()* e irá-se colocar o nome do ficheiro que encontra no campo do *Firestore* presente no campo *filename* e será adicionada a uma lista do tipo *string* e no final de percorrer todos os documentos presentes na *query* irá retornar essa lista. No final é construído um *builder* do tipo *FilterResult* e adicionado o nome dos ficheiros passando a lista retornada pelo método anterior. De seguida é passado esse *builder* no método *onNext* do *observer* que irá receber a resposta para a aplicação cliente e no final fazer *onCompleted* sobre o mesmo *observer*. Este código será realizado dentro de um *try-catch* e será apanhada ainda a exceção *ParseException* em que irá colocar no *onError* do *responseObserver* um erro criado chamado *DateFormatException* que será apanhado quando o formato das datas no pedido estiver incorreto.

4. Cliente

Com o servidor implementado, segue-se a implementação do cliente. A aplicação cliente irá chamar as funcionalidades presentes no cliente e apresentar os seus resultados numa consola.

No método *main* desta aplicação é usada uma *Cloud Function* do tipo (explicada no ponto 4.4) que irá escolher o *IP* para fazer conexão com o servidor e, após estabelecida a conexão apresenta na consola um menu com as funcionalidades do servidor (submissão de imagem, listagem de caraterísticas e filtragem de ficheiros). Após escolhida a opção é apresentada a resposta com métodos auxiliares (descritos nos pontos 4.1-4.3) que irão chamar os métodos presentes no servidor através de um *stub* não bloqueante para a primeira funcionalidade e um *stub* bloqueante para as restantes funcionalidades.

4.1 Submissão de imagem

Neste método é pedido ao utilizador que introduza o *path* do ficheiro de imagem a ser submetido, caso este não exista é apresentada uma mensagem a informar o utilizador que o *path* introduzido não existe. Caso exista é enviado ao servidor, através do método *uploadImage*, os blocos de *bytes* do ficheiro através de um *StreamObserve*r do tipo *ImageRequest* em que por cada bloco existente no ficheiro é chamado o método *onNext* do *StreamObserver* que irá entregar ao servidor o bloco de *bytes*, o nome do *blob* e o *ContentType* do ficheiro. Após o envio de todos os blocos de *by*tes é chamado o método *onComplete* do *StreamObserver* e quando o *ClientStreamObserver*, que é um *StreamObserver* do tipo *ImageResult*, estiver completo é apresentado o id do ficheiro submetido que estará presente na primeira posição da lista de resultados do *ClientStreamObserver*.

4.2 Listagem de Características

Neste método é pedido ao utilizador que introduza o id do ficheiro a apresentar as características. De seguida é enviado ao servidor um *ImageResult* com esse id através do método *getLabelsList*, que retorna um objeto do tipo *Labels* com uma lista com as características em inglês e outra com as características em português que são apresentadas na consola.

4.3 Filtragem de Ficheiros

Neste método é pedido ao utilizador que introduza uma data inicial e uma data final no formato *dd/MM/yyyy*, caso não estejam neste formato é apresentada uma mensagem a informar o utilizador bem como se a data inicial for depois da data final. Caso nenhum destes problemas aconteça, é pedido ao utilizador que introduza a característica para a filtragem. De seguida é enviado ao servidor um *FilterRequest* com os três parâmetros introduzidos pelo utilizador através do método *filterFiles* que retorna um objeto do tipo *FilterResult* com uma lista com os nomes dos ficheiros que satisfazem o filtro que são apresentadas na consola.

4.4 *Cloud Function IP Lookup*

Nesta *Cloud Function* é implementado um serviço que recebe um *HttpRequest* e um *HttpResponse* onde dado um *URL* com uma *query string* que contenha o nome de um *instance group* irá enviar todos os *IPs* presentes nesse *instance group*. Para tal é preciso usar a *API Compute* da *Google Cloud Platform* de modo a ser possível encontrar as instâncias do *instance group* recebido na *query string*. Por fim é enviado uma resposta em *JSON* através do *HttpResponse* recebido como parâmetro.

Do lado do cliente é recebida resposta e escolhido um desses *IPs* ao acaso para ser estabelecida a conexão com o servidor.

5. LabelsApp

Depois da implementação do servidor segue-se a implementação da aplicação *LabelsApp* que irá receber as informações do servidor sobre a imagem carregada para a *Cloud Storage*, através de uma subscrição ao tópico *topicworkers* do *Cloud Pub/Sub*.

Esta aplicação irá posteriormente examinar a imagem e identificar as suas características através da *Cloud Vision API* publicando-as no tópico *g12-t1d-tf-topic* para serem posteriormente lidas por uma *Cloud Function* e traduzidas para português usando a *Cloud Translation API*, sendo em seguida guardadas num documento *Firestore*.

Visto isto, para a aplicação *LabelsApp* funcionar corretamente esta necessita das dependências para *Cloud Pub/Sub* e *Cloud Vision API*.

5.1 Leitura da subscrição ao *topicworkers*

Para realizar a comunicação entre o servidor e a *LabelsApp* foi definida uma subscrição para o tópico *topicworkers* denominada de *topicworkers-sub*. Esta subscrição será a mesma para todas as instâncias da aplicação, fazendo assim com que esta trabalhe no padrão *work-queue pattern*. A sua leitura é possível através da classe *MessageReceiveHandler* que implementa o método *receiveMessage* da interface *MessageReceiver*.

Neste método obtém-se o id do *request*, o nome do *bucket* e o nome do *blob* que são recebidos na mensagem enviada pelo servidor pela ordem respetiva e separados por espaços em branco.

5.2 Deteção das Características

Após a leitura da mensagem na subscrição irá proceder-se à deteção das características passando o nome do *bucket* e do *blob* para o método auxiliar *detectLabels* da classe *DetectService* este método, com o auxílio da *Cloud Vision API* realiza a leitura e análise das características do *blob* inserido no *Cloud Storage* e recebido como parâmetro.

Este método retornará para o método *receiveMessage* uma lista de todas as características detetadas ou lança exceção do tipo *IOException()* caso não tenha sido possível analisar a imagem.

5.3 Publicação da mensagem

De volta ao método *receiveMessage* será gerada uma mensagem através da concatenação entre o id do pedido com a lista de características recebidas do método *detectLabels()*. A publicação dessa mensagem é realizada através do método *publishMessage()* da classe *PubSubService* para o tópico *g12-t1d-tf-topic* de forma semelhante à publicação realizada no servidor.

Depois da publicação da mensagem o método *receiveMessage* avisa a sua subscrição que está pronto para receber mais mensagens, dando sinal de *acknowledge* caso tenha sido executado sem exceções ou *not acknowledge* caso tenha apanhado alguma exceção.

Visto que a *Cloud Vision API* apenas retornará as características em inglês, foi necessário proceder à sua tradução. Para tal será associada uma *Cloud Function* ao tópico *g12-t1d-tf-topic* que está responsável por ler as mensagens publicadas, separando as características do id do pedido e traduzindo-as com o auxílio da *Cloud Translation API*.

Depois de efetuada a sua tradução será criado um documento na coleção *image-labels* dentro da base de dados *Firestore* que contêm o nome do ficheiro, extraído do id do pedido, o próprio id do pedido, as características originais e as traduzidas e por fim a data de criação do documento.

Para a *Cloud Function* ter acesso a estes componentes foi necessário adicionar as dependências para *Cloud Translation API* e para *Firestore* ao seu projeto.