



Faculdade Ciências Exatas e da Engenharia

Licenciatura em Engenharia Informática

Estágio de Licenciatura

API Face

Conector e Aplicação



Jénifer Constantino, 2045618

Funchal 30 de Junho de 20201

Índice

Índice	1
Introdução	2
Problema	3
Conector	3
Aplicação	3
Objetivos	4
Revisão da Literatura	4
Detecção Facial	4
Como Funciona	5
Porque Reconhecimento Facial	5
Vantagens	5
Desvantagens	6
API's	6
Evolução	6
Funcionamento	7
Tipos de API's	7
REST API	7
Face API	8
Uso de conectores	8
Funcionamento do Sistema	9
Estudo de Mercado	11
Alternativas aos conectores	11
Alternativas à Aplicação	12
Implementação	12
Requisitos - Conector	12
Requisitos - Aplicação Face Detection	14
Processo	16
Conector - Evolução da implementação	17
Aplicação - Evolução da implementação	20
Avaliação	20
Discussão	26

Introdução

O projeto a ser discutido ao longo do presente relatório foi elaborado na Connecting Software, tendo como orientadora na empresa a engenheira Filipa Nóbrega e como orientador na universidade o professor Filipe Quintal.

A **Connecting Software** é constituída por uma equipa internacional e oferece uma plataforma de integração de software, Connect Bridge, bem como SaaS e produtos locais. A empresa investe no desenvolvimento de conectores uma vez que aumenta as capacidades da plataforma de integração de software, Connect Bridge, e apresenta uma boa oportunidade de desenvolvimento para os programadores.

Esta empresa foi escolhida para a realização do projeto de licenciatura uma vez que se trata de uma empresa internacional que usa essencialmente o C# (linguagem pela qual ainda não tinha estado em contacto) e apresenta grandes capacidades de propor um projeto interessante e com conhecimento por adquirir.

Para um conhecimento mais aprofundado dos projetos da Connecting Software, são apresentados nas referências ([11] e [12]) os links para os relatórios dos alunos de mestrado em Engenharia Informática da UMA: **Vítor Batista** e **Fábio Luís**.

Numa fase inicial, a ideia recaía na realização de um conector para a API Sentiment Analysis. No entanto, após a análise da API, concluiu-se que a mesma não apresentava conteúdo suficiente para um projeto de licenciatura e, após o estudo de outras API's, proporcionadas pela Azure, conclui-se que seguiríamos pela Face API.

Assim, decidiu-se que, numa primeira fase, seria implementado um **conector** a partir da **Face API**, acompanhado pela sua **documentação** e **demo**, e, numa segunda fase, seria elaborada uma **aplicação** onde este mesmo conector pudesse ser usado. Nesta segunda fase será feita uma aplicação para a Microsoft Teams que deteta as caras das pessoas envolvidas numa chamada e, a partir de frames dessas mesmas caras, devolve as emoções que as mesmas apresentam.

Em seguida, cada um destes processos será aprofundado e detalhado, de forma a apresentar as tecnologias e requisitos abordados para cada um.

Problema

Conector

1º Problema: Hoje em dia, as empresas utilizam um grande conjunto de soluções tecnológicas para a execução de tarefas. À medida que as empresas vão crescendo, estas sentem a necessidade de utilizar diferentes softwares e estruturas que apresentam diferentes maneiras de comunicação.

Quando este problema surgiu realizava-se a integração a partir de 2 maneiras tradicionais que acabavam por não ser a melhor solução: ou por fluxo de trabalho ou por código novo elaborado pelo próprio programador, que implicava um conhecimento profundo nos sistemas a conectar e uma grande perda de tempo na elaboração deste código.

1º Solução: [1] Neste sentido, a integração de software pode ser vista como uma alternativa para quem pretende criar um fluxo de trabalho mais simples e otimizado. Com a integração de sistemas é possível que diferentes aplicações troquem dados, sejam mais produtivas e tenham um menor número de conflitos entre elas. Assim as informações circulam com maior rapidez e tornam a execução de serviços mais ágil. Com este intuito, a Connecting Software responsabiliza-se pela criação de plataformas de integração como a Connect Bridge.

No âmbito da **Connect Bridge** surge a elaboração de um conector a partir da API Face.

A Connect Bridge é uma plataforma de integração que possibilita aos utilizadores gerir dados armazenados em várias estruturas de dados, a partir da linguagem SQL.

2º Problema: A Connect Bridge fornece uma grande variedade de conectores, mas por vezes, pode acontecer, ser necessário chamar API's, serviços e sistemas em que não estão disponíveis conectores pré-construídos.

2º Solução: Para que sejam possíveis cenários mais personalizados, é permitido construir conectores customizados.

Na “Revisão de Literatura”, na parte referente aos conectores, o conceito de Connect Bridge bem como a relação com os conectores será mais aprofundada.

Aplicação

Problema: Ao fazer uma chamada online, por exemplo por Teams, muitas vezes

não é possível ter uma noção geral de quais as emoções que estiveram envolvidas. Por vezes, os intervenientes conseguem ser pouco expressivos nas suas opiniões, deixando em questão quais foram os sentimentos predominantes ao longo da chamada. Se, por exemplo, uma empresa estiver a expor um novo produto, este factor pode ser crítico, é importante conseguir saber o que é que os intervenientes sentem em relação ao produto.

Solução: Criação de uma aplicação de modo que, ao receber frames das caras dos intervenientes, consegue saber quais as suas emoções e enviar um resultado final das mesmas. Assim, a empresa facilmente consegue perceber quais as emoções geradas e o nível de certeza das mesmas.

Objetivos

O projeto tem como objetivo principal a elaboração de um **conector** que interliga a **API Face** com uma **aplicação** a ser desenvolvida para a Microsoft Teams.

No decorrer do estágio é ainda visto como objetivo a aquisição de conhecimentos relativos ao **GitHub** e às plataformas **TFS** (Team Foundation Server) e **Connect Bridge**.

Para o desenvolvimento do conector tem-se em vista a obtenção de conhecimentos na elaboração de um documento para a API Face (estudo da mesma), aprender conceitos básicos relativos a bases de dados (procedures, API Rest, CRUD operations...), obtenção de conhecimentos na linguagem de Programação Orientada a Objetos, C#, e a realização da documentação do conector desenvolvido.

Por outro lado, para o desenvolvimento da aplicação, o principal objetivo é conseguir integrar e testar o conector desenvolvido. Esta aplicação tem como objetivo principal detetar caras, mandar os seus sentimentos e o nível de certeza em relação aos mesmos.

Revisão da Literatura

Nesta parte do relatório é elaborada uma revisão literária que, numa primeira parte, consiste na exploração do conceito de **Deteção Facial**, seguindo para a **exploração** de **API's** e finalizando com a exploração ao pormenor da **API Face**.

Numa segunda parte, segue-se a contextualização do **uso de conectores**, como é que estes estabelecem uma ligação entre duas entidades, bem como as suas vantagens e desvantagens. Aqui é ainda relacionado o uso de **APIs com conectores**.

Numa última parte, é apresentado uma contextualização do **desenvolvimento de aplicações** para a **Microsoft Teams** e o porquê de se estar a recorrer a esta alternativa, acompanhado das vantagens e desvantagens desta plataforma.

Deteção Facial

O reconhecimento facial, não há muito tempo atrás, era visto como um algo de filmes, de ficção científica, no entanto, com o passar do tempo esta nova vertente tem-se tornado realidade.

Antes que possamos avançar muito mais neste assunto, vamos discutir como é que esta tecnologia de reconhecimento funciona e em que sentido foi necessário desenvolvê-la [6].

Como Funciona

Muito geralmente, a tecnologia faz scan da cara das pessoas e analisa-a. Existem 3 passos principais que todas as tecnologias de reconhecimento facial realizam:

Deteção facial -> é aqui que a cara é localizada e detectada na forma de imagens ou vídeos.

Captura Facial -> consiste na captura de características faciais. A biometria facial captura o rosto e transforma as informações faciais, capturadas em informações digitais, por meio da aplicação de um algoritmo.

Correspondência e verificação -> as informações faciais recolhidas são combinadas com as informações de outras caras, presentes na base de dados, por forma a verificar se há alguma correspondência.

Porque Reconhecimento Facial

Trata-se de um marco da tecnologia biométrica que não requer nenhuma interação do utilizador final ou um 'manipulador' para a verificar. A tecnologia é independente de qualquer processamento ou monitoramento humano. Além disso, é fácil de implementar e manter.

O reconhecimento facial pode ser aplicado em uma série de cenários. Pode ser **usado para** detectar, analisar e identificar rostos em imagens e vídeos. Com base nesta tecnologia é permitido autenticar pessoas para acesso, contar pessoas num espaço de controle de multidões ou recolher informações de multidões para campanhas de multimédia.

A deteção facial é utilizada no nosso dia a dia, desde aceder ao telemóvel a identificar amigos numa foto nas redes sociais. Às vezes esta tecnologia é ainda confiada para tomar decisões de contratar ou demitir funcionários.

Este tipo de tecnologia torna a autenticação mais rápida, simples e precisa.

Contudo a detecção facial tem trazido muita controvérsia apresentado, como qualquer outra tecnologia, **prós e contras**. Em seguida são apresentadas as vantagens e desvantagens da mesma [7]:

Vantagens

1. Aumento da segurança e proteção, prevenindo crime;
2. Ajuda a encontrar pessoas desaparecidas;
3. Protege os negócios de roubos, a partir da identificação de possíveis criminosos;
4. Requer menos recursos que outras medidas de segurança.
5. Torna o processo de compras mais fácil.
6. Melhora tratamentos médicos. A partir da detecção facial é possível identificar doenças genéticas. Em alguns casos é possível determinar como as mutações genéticas específicas causaram um determinado síndrome.

Desvantagens

1. Ameaças à privacidade;
2. Violação de direitos;
3. Restringe a liberdade;
4. Roubo de dados pessoais;
5. Fornece oportunidades para fraudes e outros crimes.
6. Possíveis erros devido à própria tecnologia.

No caso do projeto a desenvolver a detecção facial será usada no âmbito de uma aplicação para a detecção de emoções, que será implementada para a Microsoft Teams.

API's

API's (Application Programming Interface), são constituídas por pequenos trechos de código que permitem que dispositivos digitais, aplicações de software e servidores de dados se interliguem.

As API's, quando usadas, conseguem filtrar os detalhes necessários, tornando todo o processo melhor ao olho humano.

A razão para a qual as APIs existem é para oferecer suporte à integração. A integração de uma API é simplesmente a conexão entre duas (ou mais) aplicações, programas, serviços ou sistemas. As aplicações usam as APIs para enviar e receber dados e conteúdo entre si.

Evolução

O significado de API tem vindo a evoluir com o passar do tempo. Primeiro era visto como uma interface direcionada para o utilizador final, isto é, para os programas das aplicações (Program Applications). Hoje em dia, o termo API é muito mais amplo, incluindo também software utilitário e até interfaces de hardware.

A ideia de uma API é muito mais antiga que o nome e, ao longo do tempo, o conceito de API foi descoberto e não inventado.

O conceito de API já era utilizado na década de 1940 por dois cientistas que

trabalhavam em bibliotecas de software modulares. Assim, a denominação de Program Application Interface surgiu oficialmente em 1968 sendo descrito como a interação de uma aplicação (um programa gráfico) com o resto do sistema do computador. Esta foi elaborada com o intuito de libertar o programador de lidar com comportamentos peculiares do dispositivo de exibição de gráficos e fornecer independência no hardware se o computador ou a exibição fossem substituídos.

Em seguida, o termo foi introduzido no campo das bases de dados em 1974. Posteriormente, em 1990, a API foi definida simplesmente como um conjunto de serviços disponíveis para um programador para executar certas tarefas, passando as APIs a suportar todos os tipos de programação e não apenas a programação de aplicações.

A concepção de API voltou a evoluir com o surgimento das web APIs, que descreveram a ideia de uma interface de programação de aplicações baseadas em rede. As web API 's são, agora, o significado mais comum do termos API. [2]

Funcionamento

As API's, de entre outros exemplos, servem para:

1. Suportam aplicações para desktop;
2. Suportam a maioria das aplicações web;
3. Tornam possível as aplicações para telemóvel;
4. Conectar aparelhos à internet.

Assim, as API's funcionam a partir da partilha de dados e informações entre aplicações, sistemas e aparelhos, tornando possível a comunicação entre eles. Muitas das vezes as API's são explicadas pela seguinte metáfora, que se passa num restaurante:

“Um cliente fala com o empregado e pede o que pretende. O empregado escreve a ordem e comunica na cozinha. A cozinha faz o seu trabalho, criando comida, e depois o empregado devolve o pedido ao cliente”

Nesta metáfora o cliente é como um utilizador que diz o que quer. O empregado é como uma API: recebe os pedidos do utilizador e traduz o pedido em instruções fáceis de perceber pelo cozinheiro. O cozinheiro completa o pedido, através de um conjunto de regras e inputs que a cozinha facilmente reconhece. A cozinha é como um servidor que faz o trabalho de criar o pedido na maneira que o cliente quer. Quando o pedido estiver pronto, o funcionário leva-o para o cliente. [3]

Tipos de API's

Há muitos tipos de API 's: REST API, SOAP API, GraphQL API entre outras. A API a ser utilizada neste projeto (**Face API**) trata-se de uma REST API e, como tal, é este tipo de API que será falada em seguida.

REST API

REST significa “**R**epresentation **S**tate **T**ransfer” (transferência representacional de estado). É um estilo de arquitetura de software que define um conjunto de restrições a serem usadas para a criação de web services. Os web services estão em conformidade com o estilo de arquitetura REST, denominados web services RESTful.

Assim, uma **REST API** ou **RESTful API**, é um estilo de arquitetura que usa pedidos HTTP para aceder e usar os dados de uma API. Os web services RESTful permitem que os sistemas que fazem pedidos à API, possam aceder e manipular representações textuais de recursos da web usando um conjunto uniforme e predefinido de operações sem estado. Uma API RESTful usa a **metodologias HTTP** existentes para que os dados possam:

- > ser usados para recuperar um pedido - **GET**
- > alterar um estado ou atualizar um pedido que pode ser um objeto, arquivo ou bloco - **PUT**
- > criar um pedido - **POST**
- > remover um pedido - **DELETE**

Uma API para websites é um código que permite dois softwares de programas comunicarem um com o outro. A API mostra a forma mais apropriada para o developer escrever um programa, pedindo serviços de um sistema operacional ou de outra aplicação[4].

Face API

Existem várias Face API's, a usada para o projeto a ser apresentado foi a fornecida pela **Microsoft Azure**.

Esta API faz o reconhecimento facial, não implicando qualquer conhecimento de machine learning. As funcionalidades gerais incluem: detecção de rostos que reconhece os atributos numa imagem, reconhecimento de emoções interpretadas que deteta inúmeras expressões faciais como felicidade, desdém, neutralidade e medo, bem como reconhecimento e agrupamento de rostos semelhantes nas imagens.

A partir da documentação oficial desta API foi possível concluir que, apesar de não haver views, esta é constituída por **quatro entidades**: FaceList, LargeFaceList, LargePersonGroup e PersonGroup, que permitem essencialmente obter os id das listas criadas. É ainda constituída por **três funções**: ListFacesInLargeFaceList, ListPersonInLargePersonGroupPersonList e ListPersonInPersonGroupPersonList, que permitem obter o conteúdo dentro de cada uma das listas. Por fim, possui ainda **trinta procedures** que permitem a deteção facial pelo URL de uma imagem ou por stream, a deteção facial, a identificação de caras parecidas e muito mais.

No site da Microsoft Azure ([5]), são apresentados exemplos da API em funcionamento. Neste site são apresentadas funcionalidades como: reconhecimento de emoções interpretadas, verificação e detecção de rostos e muito mais. É aconselhado ao leitor do presente relatório passar por lá, por forma a analisar e a perceber quais são os inputs de cada funcionalidade e quais são os resultados das mesmas.

Uso de conectores

Conectores são elementos intermediários que permitem que seja fornecida uma interface que inicialmente não estava prevista no sistema que foi desenvolvido. Assim, o cliente faz um pedido, request, e o conector converte esse pedido para um que a outra entidade consiga interpretar. Por outras palavras, um conector é um proxy à volta de uma API que permite ao utilizador acessar os dados a partir da utilização de da linguagem Transact-SQL.

Assim um conector consegue aceder rapidamente a dados, eventos e outros recursos de outra aplicação, serviço, sistema e plataforma. É ainda possível construir fluxos de trabalho de aplicações lógicas que usam, processam e integram informações em ambientes baseados em nuvem, no local e em ambientes híbridos, muitas vezes sem ter de escrever nenhum código. [10]

O conector a ser desenvolvido neste projeto tem como intuito conectar a API Face com uma aplicação a ser desenvolvida na Microsoft Teams. Para tal é usado o Management Studio, de forma a suportar a partilha de dados.

No tópico implementação, mais precisamente no que se refere ao processo, o conector é explicado com mais pormenor, quer pela sua implementação como pelas funcionalidades e resultados que o mesmo fornece.

Funcionamento do Sistema

Para a realização de um conector é necessário ter em conta como é que o mesmo é integrado na plataforma Connect Bridge. Para tal, recorreu-se à tese de mestrado realizada pelo estudante Fábio Luís, estudante de mestrado em Engenharia Informática [12].

Relação entre a Connect Bridge e o Connector

A Connect Bridge Platform (CB) recorre aos conectores para permitir que os usuários se conectem a uma diversidade de sistemas. Estes conectores, recorrem às APIs para ter acesso a dados e realizar trocas de informações. A imagem apresentada abaixo foi retirada da tese acima referida [12]. Esta apresenta de forma simples, os os elementos da plataforma em questão:



Fig 1. Constituição da CB

Assim, o projeto a ser elaborado consiste na criação de um conector que vai ser integrado nesta plataforma. O conector é elaborado num IDE e, por forma a fazer com que a Connect Bridge reconheça o mesmo, o código deve estar numa pasta, reconhecida pela plataforma, encarregando o servidor de criar o conector.

Estes conectores podem ser elaborados em linguagem como o C# que por sua vez comunicam com sistemas alvo a partir de linguagem SQL, que depois é convertida em chamadas API, por forma a evitar um acesso direto à base de dados.

////POR ESQUEMA QUE TENHO DE PERGUNTAR AQUI

1. Do connector à App
2. Como é que os procedures and tables e views se relacionam

A API é constituída por operações e entidades, que, no Conector, correspondem a *procedures* e tabelas, respetivamente.

Aplicações para a Microsoft Teams

A Microsoft Teams apresenta ferramentas e processos confiáveis para os quais, cada vez mais, há mais pessoas a aderir a esta tecnologia. Para além disto, de forma a conseguir uma melhor modelação às necessidades dos clientes, a Microsoft Teams permite ainda a criação e integração de novas aplicações.

Porquê usar a Microsoft Teams?

A Microsoft Teams é muito mais que apenas uma ferramenta de chat ou de reuniões. Esta apresenta muitas outras ferramentas que ajudam no fluxo de trabalho. Como tal em seguida é indicado brevemente algumas das suas funcionalidades e, consequentemente, algumas das suas vantagens:

1. Permite a criação de um chat entre duas pessoas ou um grupo de pessoas. Facilita a transferência de ficheiros.
2. Permite uma boa preparação para as reuniões e uma boa organização pós reuniões.
3. Melhora a colaboração e comunicação. Graças à integração total com o Office 365 é possível partilhar um documento Word, Excel ou PowerPoint e

editá-lo em simultâneo.

4. Permite manter estar em contacto em qualquer altura e em qualquer lugar.

Existem **vários tipos** de aplicações para a Microsoft Teams:

Personal apps: uma página ou bot dedicado para ajudar os utilizadores a concentrarem-se nas suas próprias tarefas)

Tabs: exibe o conteúdo da uma página web numa tab onde as pessoas podem discutir e trabalhar juntas)

Bots: permite iniciar fluxos de trabalho dentro do Teams, como gerar um pedido, rever código...)

Messaging extensions: permitem compartilhar rapidamente informações externas em uma conversa, bem como criar um tíquete de ajuda com base no conteúdo de um post num dado canal)

Meeting extensions: aplicações relacionadas a chamadas no Teams

Webhooks and connectors: são uma maneira simples de enviar notificações automaticamente de outra aplicação para um canal no Tams

Microsoft Graph for Teams: fornece acesso a informações sobre equipas, canais, utilizadores e mensagens que podem ajudar a criar ou aprimorar recursos numa dada aplicação). [14]

Com isto explicado, passou-se ao estudo de mercado onde foi possível analisar aplicações ou conceitos semelhantes aos conectores, conseguindo obter uma comparação com os sistemas a serem desenvolvidos e em que sentido é que estes se podem destacar no mercado de trabalho.

Estudo de Mercado

Nesta parte do relatório é apresentado o resultado de uma pesquisa relativa à existência de aplicações ou serviços no mercado semelhantes ao conector e à aplicação a ser desenvolvida.

Alternativas aos conectores

Como dito na apresentação do problema que levou a elaboração do projeto corrente, antes do aparecimento da plataforma Connect Bridge, a criação de soluções de integração era feita principalmente por meio de código feito de raiz ou por fluxo de trabalho. No entanto, elaborar código específico para a conexão entre dois sistemas implica ter um bom conhecimento de ambos e acaba por consumir bastante tempo. Nenhuma destas soluções tradicionais acabam por ser uma boa escolha.

Assim, para além dos conectores produzidos pela Connecting Software, existem outras ideias semelhantes que vêm resolver o mesmo problema, uma delas é a **IFTTT: If This Then That** [13].

O IFTTT é uma plataforma que integra diversos serviços que interagem entre si de forma automática e, por isso, sem a intervenção humana. Alguns exemplos de serviços integrados no IFTTT são dispositivos para casas inteligentes, Spotify, Uber entre outros.

Assim, tal como indicado acima, o nome do sistema é *“If this then that”*. Assim, quando acontece algo definido pelo utilizador, o sistema vai provocar uma determinada ação também definida pelo utilizador.

Por exemplo, é possível utilizar a localização do telemóvel como um gatilho, para que, quando o utilizador saia de uma determinada área, uma lâmpada de casa se ligue ou desligue.

Outro exemplo é que, ao detetar que a bateria do telefone está com uma certa percentagem é mandada automaticamente uma mensagem a alguém.

Assim é possível estabelecer a ligação entre sistemas a partir de um gatilho provado por um deles que vai ter uma consequência no outro.

A **semelhança** entre o IFTTT e os conectores é que ambos responsabilizam-se por criar uma ligação entre 2 sistemas que à partida não estavam preparados para interagirem.

No entanto, uma das características que os **distingue** é a maneira e o fim para que são utilizados. No caso do IFTTT este não se trata de uma plataforma ou meio de integração, mas sim de **interação** entre aplicações. Para além de que esta plataforma funciona por triggers, ao passo que os conectores, para além de integrarem uma aplicação noutra, funcionam, não por triggers, mas por pedidos.

Alternativas à Aplicação

Como anteriormente referido, a aplicação consiste em, numa reunião do Teams, retirar frames da cara das pessoas e detetar as suas emoções, de forma que, no final de uma reunião seja possível saber quais as emoções predominantes.

Após realizar algumas pesquisas, foi possível concluir que, apesar de haver aplicações que fazem a detecção facial e de emoções, nenhuma delas está desenvolvida para o Teams. Exemplo de aplicações ou serviços desenvolvidos no âmbito da detecção facial e, consequentemente de emoções, são **iMotions**, **Veriff** e **Emotimeter - Emotion detector**.

Implementação

Em seguida são apresentados os requisitos, a evolução da implementação e alguns exemplos de funcionamento para o conector e para a aplicação de detecção de emoções numa videochamada no Teams.

Requisitos - Conector

Antes de começar a implementar o conector, foram levantados alguns requisitos que foram evoluindo ao longo do tempo. Estes requisitos descrevem características que o conector terá que cumprir.

Em seguida são apresentados requisitos de **negócio**, **tecnologia**, **funcionais**, **não funcionais** e requisitos de **dados**. É de apontar que, uma vez que o conector estabelece a conexão entre dois sistemas (a API e a aplicação da Microsoft Teams), não foram definidos requisitos de utilizador.

À frente de cada requisito é ainda colocada qual foi a fonte pela qual foi possível obter o requisito.

Requisitos de Negócio:

Este tipo de requisitos são restrições aplicadas a uma ação comercial de alguma empresa, de maneira a atender ao negócio e funcionar conforme o esperado e as regras estabelecidas.

R.N.1. “O conector a ser desenvolvido deve aumentar as capacidades da plataforma de integração de software, Connect Bridge.” **Fonte:** Connecting Software

R.N.2. “O conector deverá fornecer uma boa oportunidade de desenvolvimento para os programadores.” **Fonte:** Connecting Software

Requisitos de Tecnologia:

Este tipo de requisitos estipulam a base para o desenvolvimento de software e os comportamentos que o mesmo deve apresentar.

R.T.1. “O IDE a ser utilizado para o desenvolvimento deverá ser o Visual Studio.” **Fonte:** Connecting Software

R.T.2. “A linguagem de programação a ser usada para o desenvolvimento do conector deverá ser o C#.” **Fonte:** Connecting Software

R.T.3. “A API a ser utilizada deverá ser a Face API da Microsoft Azure.” **Fonte:** Connecting Software

Requisitos Funcionais:

Este tipo de requisitos define qualquer funcionalidade que o conector deverá apresentar.

R.F.1. “O conector deverá ser capaz de receber frames de imagens.” **Fonte:** Pesquisa

R.F.2. “O conector deverá fazer pedidos à API Face.” **Fonte:** Pesquisa

R.F.3. “O conector deverá permitir a ligação entre a API Face e a aplicação a ser desenvolvida para a Microsoft Teams.” **Fonte:** Pesquisa

R.F.4. “O conector deverá permitir à Microsoft Teams enviar frames de caras.” **Fonte:** Pesquisa

R.F.5.“O conector deverá permitir à Microsoft Teams obter as emoções detectadas num frame de vídeo.” **Fonte:** Pesquisa

R.F.6.“O conector deverá permitir à Microsoft Teams obter o nível de confiança na detecção de uma emoção.” **Fonte:** Pesquisa

R.F.7.“O conector deve respeitar os protocolos definidos na documentação oficial da Microsoft API Face definidos no seguinte link: [Face REST API reference - Azure Cognitive Services | Microsoft Docs](#)” **Fonte:** Pesquisa

Requisitos não Funcionais:

Este tipo de requisitos vem cobrir a exigência técnica de um software, nomeadamente aspetos de segurança do sistema, desempenho, ou até mesmo, prevenção de possíveis falhas.

R.N.F.1. “O conector deverá ser capaz de estabelecer uma ligação com a conta da Azure.” **Fonte:** Connecting Software

R.N.F.2. “O conector deverá ser capaz de estabelecer uma ligação segura com a plataforma Connect Bridge.” **Fonte:** Connecting Software

R.N.F.3. “O sistema deverá ser capaz de receber imagens no formato JPEG, PNG e GIF.” **Fonte:** Pesquisa

Requisitos de Dados:

Estes requisitos estão relacionados com segurança, regras e consistências de dados.

R.D.1. “O conector deverá aceitar com endpoint o definido na subscrição da Azure.” **Fonte:** Pesquisa

R.D.2. “O conector deverá aceitar como ‘subscription key’ a definida pela Azure.” **Fonte:** Pesquisa

R.D.3. “O conector deverá respeitar os requisitos de dados estipulados pela documentação oficial da Microsoft API Face, definidos em: [Face REST API reference - Azure Cognitive Services | Microsoft Docs](#)” **Fonte:** Pesquisa

Requisitos - Aplicação Face Detection

Após elaborar o conector começou-se a estudar quais seriam os requisitos para a aplicação a ser elaborada na Microsoft Teams.

Requisitos de Negócio:

Este tipo de requisitos são restrições aplicadas a uma ação comercial de alguma empresa, de maneira a atender ao negócio e funcionar conforme o esperado e as regras

estabelecidas.

R.N.1. “A aplicação a ser desenvolvida deve permitir melhorar o feedback em relação a um produto.” **Fonte:** Connecting Software

Requisitos de Tecnologia:

Este tipo de requisitos estipulam a base para o desenvolvimento de software e os comportamentos que o mesmo deve apresentar.

R.T.1. “A aplicação deverá ser desenvolvida para a Microsoft Teams.” **Fonte:** Connecting Software.

R.T.2. “A aplicação deverá utilizar o conector desenvolvido.” **Fonte:** Connecting Software.

R.T.3. “A aplicação deverá ser desenvolvida em C# ” **Fonte:** Connecting Software.

R.T.4. “O IDE a utilizar será o Visual Studio.” **Fonte:** Connecting Software.

R.4.5. “Considerando o tipo de aplicações possíveis na Microsoft Teams, a aplicação a ser desenvolvida deverá ser uma Tab e uma Meeting extension. ” **Fonte:** Pesquisa

Requisitos Funcionais do Sistema:

Este tipo de requisitos define qualquer funcionalidade que o conector deverá apresentar.

R.F.1. “O sistema deverá detetar caras numa video-chamada.” **Fonte:** Connecting Software

R.F.2. “O sistema deverá enviar frames de partes da chamada para o conector.” **Fonte:** Pesquisa

R.F.3. “O sistema deverá enviar parâmetros de acordo com a escolha do utilizador.”

R.F.3.2. “O sistema deverá permitir ao utilizador escolher quais os atributos faciais que pretende receber de um dado interveniente na chamada.” **Fonte:** Connecting Software

R.F.3.1. “O sistema deverá permitir escolher ver a idade dos intervenientes. ” **Fonte:** Pesquisa

R.F.3.2. “O sistema deverá permitir escolher ver os acessórios utilizados pelos intervenientes. ” **Fonte:** Pesquisa

R.F.3.3. “O sistema deverá permitir escolher ver o nível de blur numa chamada” **Fonte:** Pesquisa

R.F.3.4. “O sistema deverá permitir escolher ver a **emoção** dos intervenientes numa chamada.” **Fonte:** Pesquisa

R.F.3.5. “O sistema deverá permitir escolher ver o nível de exposição de uma vídeo chamada: alto, baixo ou médio.” **Fonte:** Pesquisa

R.F.3.6. “O sistema deverá permitir escolher ver informações sobre a existência de pelo facial: com barba, bigode e patilhas. ” **Fonte:** Pesquisa

R.F.3.7. “O sistema deverá permitir escolher ver o género dos intervenientes na chamada.” **Fonte:** Pesquisa

R.F.3.8. "O sistema deverá permitir escolher ver o tipo de óculos que os intervenientes na chamada utilizam: óculos de sol, óculos de leitura, óculos de natação ou sem óculos." **Fonte:** Pesquisa

R.F.3.9. "O sistema deverá permitir escolher ver informações sobre o cabelo dos intervenientes na chamada: careca, cor do cabelo e a visibilidade do cabelo na imagem." **Fonte:** Pesquisa

R.F.3.10. "O sistema deverá permitir escolher ver informações relativas à posição da cabeça dos intervenientes na chamada." **Fonte:** Pesquisa

R.F.3.11. "O sistema deverá permitir escolher ver informações relativas à existência de maquilhagem dos intervenientes: existência de maquilhagem nos olhos e nos lábios." **Fonte:** Pesquisa

R.F.3.12. "O sistema deverá permitir escolher ver informações relativas ao nível de barulho, isto é, de informação na chamada realizada." **Fonte:** Pesquisa

R.F.3.13. "O sistema deverá permitir escolher ver informações relativas à obstrução de um dado rosto: olho, testa ou boca ocultos." **Fonte:** Pesquisa

R.F.3.14. "O sistema deverá permitir escolher ver a intensidade, entre 0 e 1, do sorriso de um dado interveniente." **Fonte:** Pesquisa

Requisitos de Utilizador/utilização:

Descrevem objetivos ou tarefas que devem ser possíveis conseguir com o sistema.

R.U.1. "O sistema deverá permitir ao utilizador obter as emoções mais predominantes numa chamada."

R.U.2. "O sistema deverá permitir ao utilizador saber a emoção de um dado utilizador."

Requisitos não Funcionais:

R.N.F.1. "O sistema deverá estar disponível a maior parte do tempo."

R.N.F.2. "O sistema deverá ser executado na plataforma Microsoft Teams."

R.N.F.3. "O sistema deverá obedecer às leis definidas pelo estado."

R.N.F.4. "O sistema deverá comunicar com o conector."

R.N.F.5. "O sistema deverá ser capaz de detectar até 100 caras numa videochamada"

R.N.F.6. "O sistema apenas envia frames para o conector em formato JPEG, PNG e GIF."

Processo

Para o desenvolvimento do projeto em questão os objetivos principais recaíram na elaboração de um conector, acompanhado por documentação e, por fim, de uma aplicação para o Microsoft Teams.

Ao longo de todo o processo de desenvolvimento houveram alterações e o planeamento inicial do que seria feito ao longo do estágio teve de passar por algumas mudanças.

A ideia inicial era elaborar um conector a partir da Text API, no entanto, com alguma pesquisa concluímos que não apresentava conteúdo suficiente para a elaboração de um projeto de licenciatura.

Com isto, foi feita uma pesquisa e, com a aprovação da empresa, concluímos que a melhor API a ser utilizada seria a Face API. Decidimos que seria melhor implementar alguns procedures e tabelas que permitissem a deteção de emoções e, em seguida, implementar uma aplicação para a Microsoft Teams.

Ao elaborar os procedures inicialmente indicados, observou-se que estavam dependentes de outros que não estavam no plano. Como tal e, após falar com a orientadora na empresa, concluímos que a melhor solução seria implementar todos os procedures, tabelas e funções que a API apresentava.

Com o conector finalizado, foi feita a documentação e uma demo do mesmo.

Por fim, era pretendido construir uma aplicação para o teams, no entanto, por falta de tempo não foi possível realizar a mesma.

Para a elaboração do conector, tive a oportunidade de ser introduzida a duas plataformas muito utilizadas pela empresa:

O TFS - Team Foundation Service, onde seriam colocadas as minhas '*user stories*' para as quais teria de implementar num dado intervalo de tempo que seria estipulado nesta ferramenta. A partir do TFS era ainda possível colocar o estado de uma tarefa: se estava em progresso, acabada ou por fazer. É de referir que esta ferramenta acabou por não ser muito utilizada. Uma vez que eu demorava muito tempo a desenvolver uma dada funcionalidade e acabava por não proceder aos tempos estipulados, acabando por não se mostrar útil e caiu em desuso no projeto.

Outra ferramenta utilizada foi o GitHub. Esta ferramenta, ao contrário da anterior, acompanhou o desenvolvimento do projeto na totalidade. Foi-me introduzido alguns conceitos básicos relativos ao mesmo e rapidamente aprendi funcionalidades básicas que foram utilizadas ao longo do projeto, como fazer commits, push, pull, ver o histórico, entre outras funcionalidades básicas.

Em seguida é explicado o processo da elaboração do conector, acompanhado com o processo inicial que era esperado e tudo aquilo que foi alterado ao longo do desenvolvimento.

Conector - Evolução da implementação

Plano inicial

- Levantamento de requisitos funcionais
 - Explorar a API
 - Mapear entidades da API para entidades SQL
 - Compreender se as entidades da API são estáticas (não mudam) ou dinâmicas (podem mudar)
- Definir um plano; Ser introduzida ao TFS/GIT e ao processo interno da empresa
- Iniciar o desenvolvimento do conector, tendo em consideração a criação de testes

unitários

- Definir as propriedades usadas para estabelecer uma ligação ao sistema destino (i.e. a API)
- Validar a ligação
- Definir o modelo SQL (i.e. *tables, functions, stored procedures, views*)
- Implementar a execução de operações CRUD
- Converter o resultado da API em uma tabela
- Desenvolvimento de testes de integração
- Documentar o conector
 - *Configuration Guide* – define as propriedades do conector e explica como criar uma conta no CBMS
 - *QuickStart Guide* – oferece alguns exemplos de utilização do conector, incluindo uma aplicação de integração com outro sistema (utilizando outro conector)
 - *Reference Guide* – define o modelo SQL (documento gerado automaticamente e exige que o conector tenha definido descrições para colunas, parâmetros, etc.)

Abordagem utilizada

Como é possível observar pelo plano, ao longo do projeto foi realizada uma abordagem em **cascata**. Foi feito o levantamento inicial dos requisitos funcionais, para os quais o conector deveria de implementar, e foi elaborado um documento com todas as informações da AP [].

Como qualquer abordagem em cascata, houveram naturalmente algumas alterações nos planos propostos inicialmente, no entanto, esta parte inicial de levantamento de requisitos e estudo da API revelou-se fundamental para que fosse possível desenvolver o conector com mais confiança, conseguindo coordenar melhor fatores de tempo.

Problemas relativos à API escolhida

Inicialmente o objetivo era utilizar a API Text Analytics, no entanto, ao realizar o levantamento de requisitos e ao elaborar o documento acima referido, observamos que a API em causa não apresentava conteúdo suficiente para a elaboração de um projeto de licenciatura.

Como tal, procedeu-se ao estudo de outras APIs que fossem benéficas para a empresa. Após observar algumas APIs oferecidas pela Azure, foi escolhida a API Face. Posto isto, realizou-se novamente o levantamento de requisitos funcionais e, a partir da realização de um documento [**Colocar em anexo**], o estudo da API.

Ferramentas usadas

Para além das mencionadas acima (GitHub e TFS), foram utilizados o Visual Studio, o Conncet Bridge Management Studio, CBMS, e a plataforma Azure. Para CBMS é utilizada uma linguagem Transact SQL.

Todas estas plataformas foram brevemente introduzidas antes do desenvolvimento do conector.

Elaboração do Conector

Como primeiro passo e, como enunciado em cima, comecei por definir as propriedades que ligam o sistema à API (Endpoint e Subscription Key), criei um cliente, uma sessão e fiz a ligação entre o conector e o Connect Bridge Management Studio.

Com isto feito, o plano seria implementar os procedures, as tabelas e as funções necessárias, bem como os **testes unitários** relevantes para cada uma delas. Definiu-se que:

Procedures necessários:

1. DetectFaceWithURL
2. DetectFaceWithStream
3. FindSimilar
4. GroupFacesBySimiliarity
5. IdentifySimilarities
6. VerifyFaceToFace
7. VerifyFaceToPerson
8. TrainLargeFaceList
9. GetTrainingStatus

Tabelas necessárias:

1. FaceList
2. LargeFaceList

Funções necessárias:

1. ListFacesInFaceList

Perante isto, decidi começar a implementar o código e os testes unitários para os procedures, tabelas e funções necessárias, e só depois testar no CBMS.

Ao testar no CBMS cheguei à conclusão que haviam procedures, tabelas e funções em falta e que não se encontravam no plano para serem implementadas. Por exemplo, o procedure VerifyFaceToPerson precisava como parametros de um PersonId (obtido pelo procedure não implementado CreatePersonGroupPerson) e de um LargePersonGroupId (obtido pela tabela não implementada: LargePersonGroup).

Perante isto, com o acompanhamento da orientadora, decidimos que a melhor opção seria implementar a API Face toda, uma vez que, todos os procedures/ funções / tabelas tinham dependências uns com os outros. Para além dos mencionados em cima, passamos a implementar os seguintes procedures, tabelas e funções:

Procedures:

1. AddFaceFromStreamToFaceList
2. AddFaceFromURLToFaceList
3. DeleteFaceFromFaceList

4. AddFaceFromStreamToLargeFaceList
5. AddFaceFromURLToLargeFaceList
6. DeleteFaceFromLargeFaceList
7. GetTrainingStatusOfLargePersonGroup
8. ListLargePersonGroup
9. TrainLargePersonGroup
10. AddFaceFromStreamToLargePersonGroupPerson
11. AddFaceFromURLToLargePersonGroupPerson
12. CreateLargePersonGroupPerson
13. DeleteLargePersonGroupPerson
14. UpdateLargePersonGroupPerson
15. GetTrainingStatusOfPersonGroup
16. TrainPersonGroup
17. AddFaceFromStreamToPersonGroupPerson
18. AddFaceFromURLToPersonGroupPerson
19. CreatePersonGroupPerson
20. DeletePersonGroupPerson
21. UpdatePersonGroupPerson

Tabelas:

1. LargePersonGroup
2. PersonGroup

Funções:

1. ListFacesInLargeFaceList
2. ListPersonInLargePersonGroup
3. ListPersonInPersonGroupPerson

Após realizar o código e os testes unitários de cada um dos procedures, funções e tabelas foi possível testar no CBMS. Houve, naturalmente, erros que foram resolvidos até que tudo funcionasse corretamente.

Foram colocados, na secção 'Avaliação', alguns screenshots que refletem exemplos de resultados obtidos com a implementação do conector.

É de lembrar que, no **Anexo x** está presente a descrição de cada procedure, tabela, função e qualquer outro elemento da API Face que foi utilizado para elaborar o conector em questão.

Como estipulado no planeamento acima, com este feito alcançado, foi realizada a documentação e uma demo do conector, no entanto, por falta de tempo não foi possível realizar os testes de integração.

Aplicação - Evolução da implementação

Por falta de tempo não foi possível realizar a aplicação para a Microsoft Teams, pelo que, esse seria o próximo passo numa futura continuação do projeto descrito no

presente relatório.

Avaliação

Para a avaliação do sistema, inicialmente, tínhamos como objetivos a elaboração de testes unitários, testes de integração e testes no CBMS (Connect Bridge Management Studio).

No entanto, devido à falta de tempo, optou-se por descartar os testes de integração, elaborando apenas testes unitários, conjuntamente com a elaboração do código para os procedures, tabelas e funções. Uma vez que estes estivessem a correr como desejado, todas estas entidades eram testadas no CBMS.

Logo em seguida podem ser observados alguns screenshots do programa em funcionamento:

Contextualização: Primeiro é criado um LargePersonGroup, a partir de um Insert na tabela LargePersonGroup. Em seguida, por forma a verificar que o Insert foi bem realizado, efetuou-se um select da tabela LargePersonGroup. Após a criação de uma large person group, foi criada uma pessoa dentro dessa tabela - **CreateLargePersonGroupPerson**. Tendo agora uma pessoa é necessário atribuir-lhe uma cara, fornecida por URL - **AddFaceFromURLToLargePersonGroupPerson**. Posto isto é detectada uma cara por URL. Este procedure é usado a fim de conseguir realizar futuros procedures - **DetectFaceWithURL**. Em seguida, é adicionada essa cara a uma large face list - **AddFaceFromURLToLargeFaceList**. Esta large face list é gerada pela tabela respectiva. Por forma a conseguir realizar futuros procedures procedeu-se ao treinamento da largeFaceList - **TrainLargeFaceList** - e obtido o seu estado - **GetTrainingStatus** (treinamento bem sucedido ou não). Apesar de não ser apresentado abaixo (uma vez que se realizava da mesma maneira), é preciso igualmente treinar o personGroup e o largePersonGroup para que seja possível testar procedures como IdentifySimilarities. Com isto feito é possível testar os procedures: **FindSimilar, GroupFacesBySimilarity, IdentifySimilarities, VerifyFaceToFace** e **VerifyFaceToPerson** de forma adequada, uma vez que, pelo menos, a large face list contém caras e a large person group contém uma pessoa com uma cara associada.

De entre várias funções implementadas, um exemplo é a ListFacesInLargeFaceList. Esta função oferece auxílio para saber quais são as caras que estão presentes dentro de uma dada Large Face List.

Exemplos de Tabelas:

As tabelas são as componentes responsáveis pelas operações CRUD - Create, Read, Update e Delete. Em seguida são apresentadas as operações de Insert (Create) e Select (Read) na tabela LargePersonGroup.

1. Insert em LargePersonGroup

Operação responsável por inserir um tuplo na tabela LargePersonGroup. Não retorna nenhum valor, para ver se realmente inseriu seria necessário realizar um select como o indicado logo de seguida.

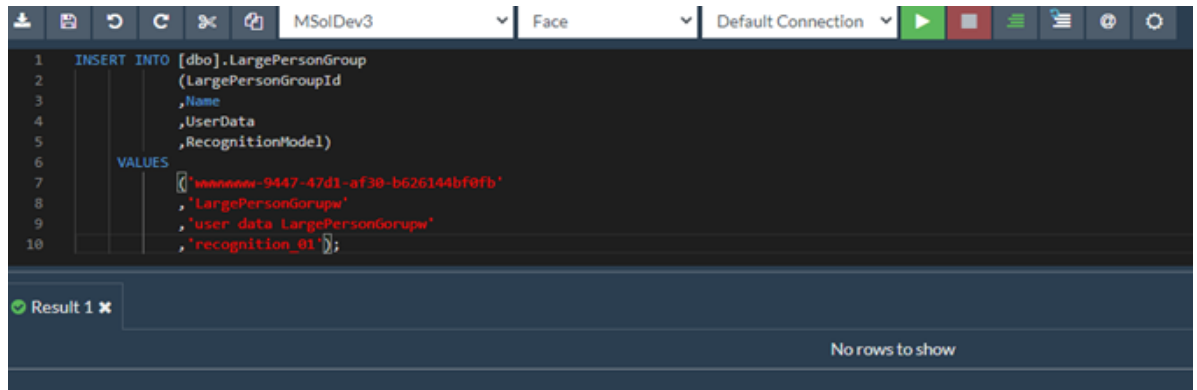


Fig 2. Exemplo de funcionamento da tabela LargePersonGroupPerson.

2. Select em LargePersonGroup

Podemos observar que a lista adicionada em cima está apresentada abaixo.

The screenshot shows a SQL query window with the following text:

```

1 SELECT [LargePersonGroupId]
2     , [Name]
3     , [UserData]
4     , [RecognitionModel]
5 FROM [dbo].[LargePersonGroup];
  
```

Below the query, the 'Result 1' tab is visible, showing a table with the following data:

LargePersonGroupId	Name	UserData	RecognitionModel
6666666-f0e3-4695-9831-ef9124d73d9d	UpdatedLargePersonGroupList	UpdatedLargePersonGroupList 1	recognition_01
7777777-f0e3-4695-9831-ef9124d73d9d	LargePersonGroup3	LargePersonGroup3 UserData	recognition_03
8888888-f0e3-4695-9831-ef9124d73d9d	LargePersonGroup2	LargePersonGroup2 UserData	recognition_02
xxxxxxxx-9447-47d1-af30-b626144bf0fb	LargePersonGroupw	user_data LargePersonGroupw	recognition_01

Fig 3. Exemplo de funcionamento da tabela LargePersonGroup.

Exemplos de Procedures:

1. CreateLargePersonGroupPerson

Procedure responsável por criar uma pessoa em um Large Person Group. Recebe como parâmetro um *LargePersonGroupId*, gerado pela tabela Large Person Group, um *nome* e um *userdata*.

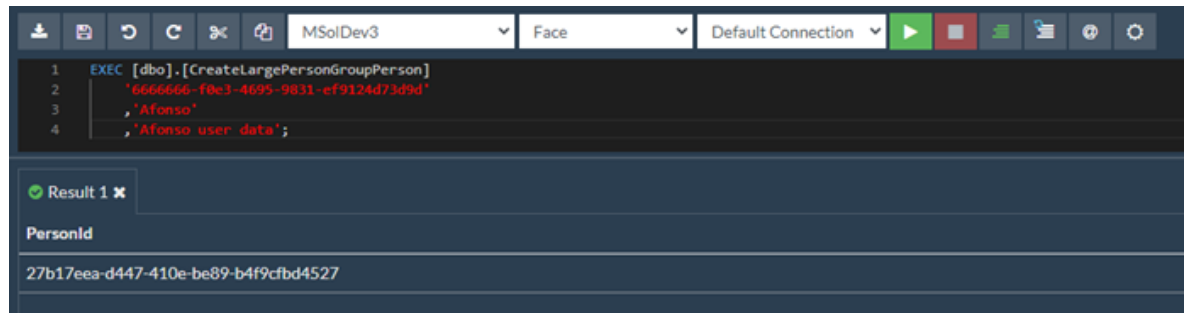


Fig 4. Exemplo de funcionamento do procedure CreateLargePersonGroupPeson.

2. AddFaceFromURLToLargePersonGroupPerson

Adiciona uma cara a uma pessoa dentro de um Large Person Group.

Recebe como parâmetros um *LargePersonGroupId*, gerado pela tabela Large Person Group, um *PersonId*, gerado pelo CreateLargePersonGroupPerson, um *detectionModel* (detection_01, detection_02 e detection_03), um *targetFace*, *userdata* e um *ImageURL*.

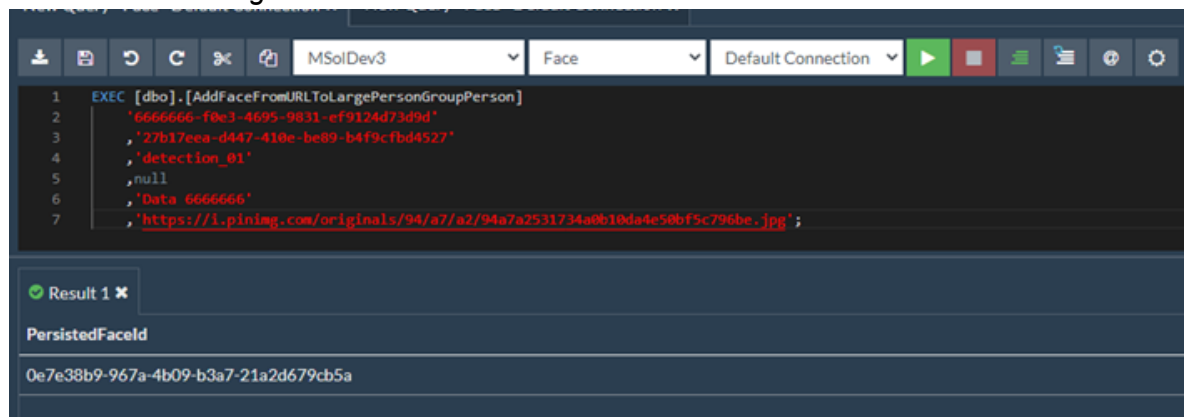


Fig 5. Exemplo de funcionamento do procedure CreateLargePersonGroupPeson.

3. DetectFaceWithURL

Podemos observar que, ao inserir uma imagem por URL, de acordo com os parâmetros inseridos, é possível ver a localização do retângulo e, consequentemente da cara, o atributo idade (também definido como parâmetro) e o faceld atribuído.

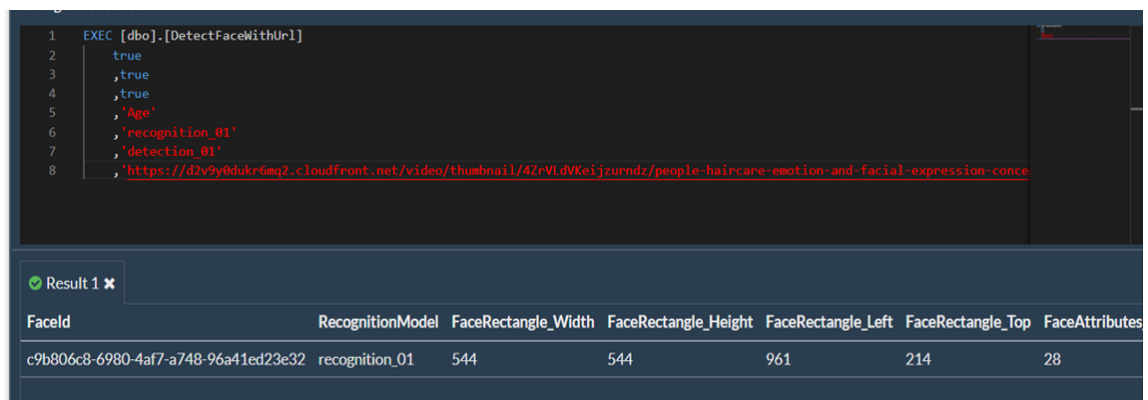


Fig 6. Exemplo de funcionamento do procedure DetectFaceWithURL.

4. AddFaceFromURLToLargeFaceList

Recebe como parâmetros um *LargeFaceListId*, gerado pela tabela *LargeFaceList*, um *userData*, um *targetFace* (usado quando a imagem contem mais que uma cara), um *detectionModel* (detection_01, detection_02 ou detection_03) e um *imageURL*.

Este procedure responsabiliza-se por colocar uma cara numa Large Face List.

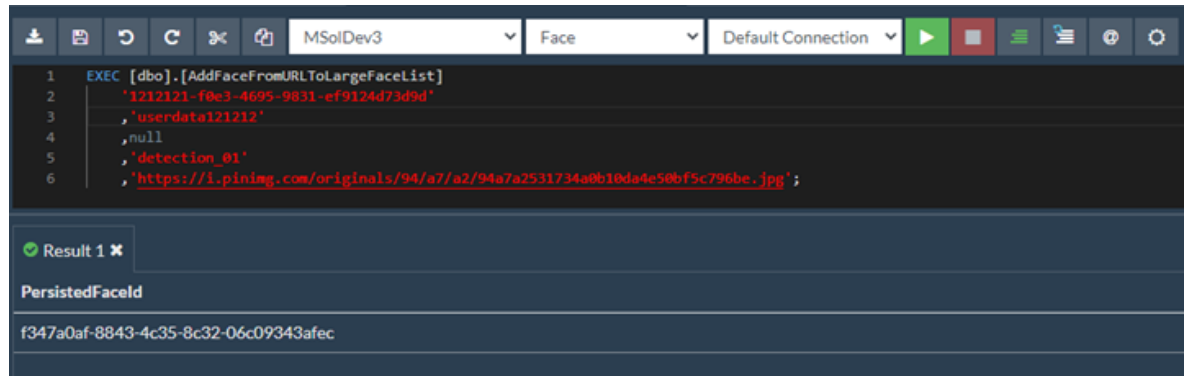


Fig 7. Exemplo do procedure AddFaceFromUriToLargeFaceList.

5. TrainLargeFaceList

Recebe como parâmetros uma large face list, criada na tabela correspondente.

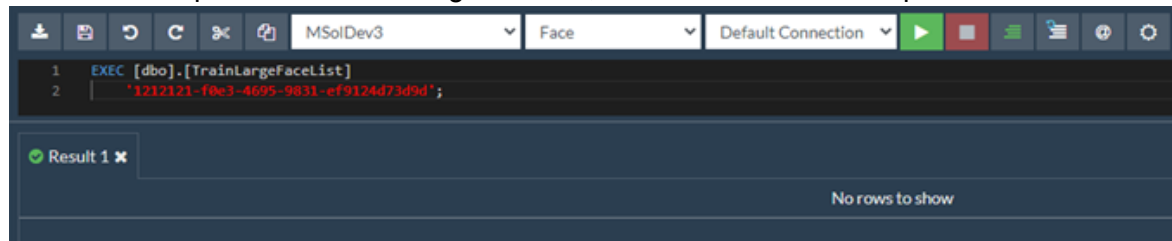


Fig 8. Exemplo de funcionamento do procedure TrainLargeFaceList.

6. GetTrainingStatus

Para que funcione de forma adequada é necessário que o id fornecido corresponda a uma large face list que tenha sido treinada. Podemos observar que o status é Succeeded e, como tal, a lista encontra-se treinada.

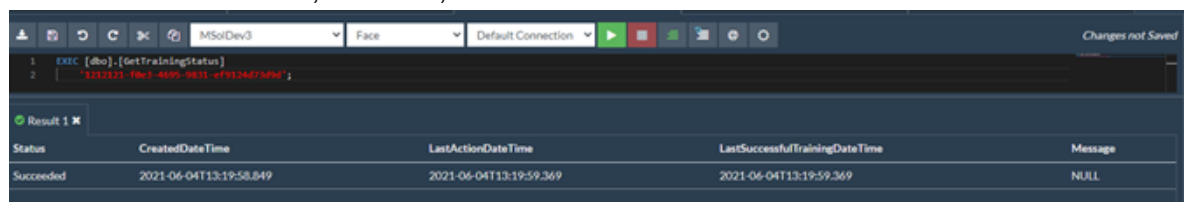
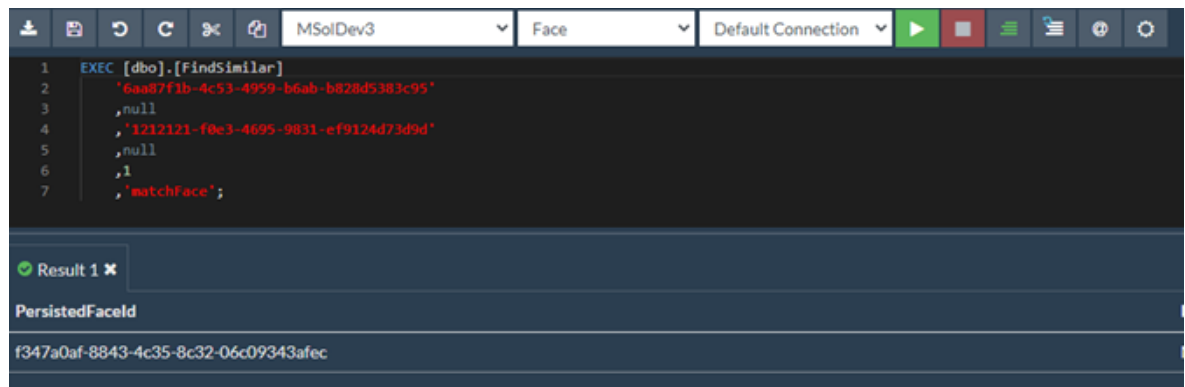


Fig 9. Exemplo de funcionamento do procedure GetTrainingStatus.

7. FindSimilar

Recebe como argumentos um *FaceId*, criado em **AddFaceFromURLToLargeFaceList** e uma das 3 opções: *FaceListId*, *LargeFaceListId* ou um array de face ids. Para além deste parâmetros recebe ainda um número máximo de candidatos que serão retornados e um modo que pode ser *matchFace* ou *matchPerson*.

Para o caso em exemplo, optou-se por escolher uma *largeFaceList* uma vez que esta já tinha sido treinada anteriormente. Foi gerada uma nova cara pelo procedure: **DetectFaceFromURL**.



The screenshot shows the SQL Server Enterprise Manager interface. The top bar indicates the server is 'MSolDev3', the database is 'Face', and the connection is 'Default Connection'. The query window contains the following T-SQL code:

```
1 EXEC [dbo].[FindSimilar]
2     '6aad7f1b-4c53-4959-b6ab-b828d5383c95'
3     ,null
4     ,'1212121-f0e3-4695-9831-e9124d73d9d'
5     ,null
6     ,1
7     ,'matchFace';
```

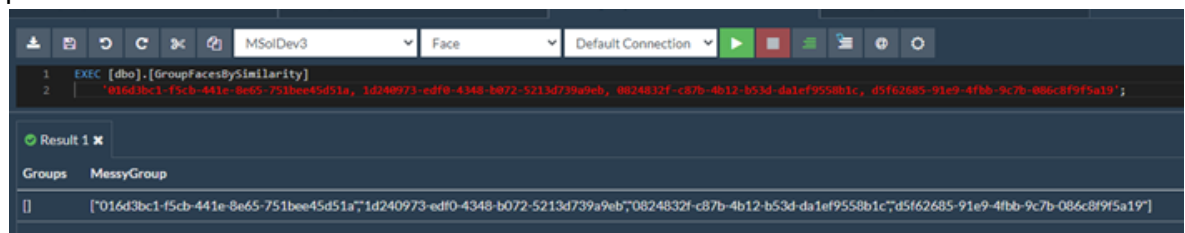
The results pane shows 'Result 1' with a single row of data:

PersistedFaceId
f347a0af-8843-4c35-8c32-06c09343afec

Fig 10. Exemplo de funcionamento do procedure FindSimilar.

8. GroupFacesBySimilarity

Para que seja possível testar este procedure é necessário gerar várias caras pelo procedure: **DetectFaceWithURL**.



The screenshot shows the SQL Server Enterprise Manager interface. The top bar indicates the server is 'MSolDev3', the database is 'Face', and the connection is 'Default Connection'. The query window contains the following T-SQL code:

```
1 EXEC [dbo].[GroupFacesBySimilarity]
2     '016d3bc1-f5cb-441e-8e65-751bee45d51a, 1d240973-edf0-4348-b072-5213d739a9eb, 0824832f-c87b-4b12-b53d-da1ef9558b1c, d5f62685-91e9-4fbb-9c7b-086c8f9f5a19';
```

The results pane shows 'Result 1' with a single row of data:

Groups
MessyGroup

Fig 11. Exemplo de funcionamento do procedure GroupFacesBySimilarity.

9. IdentifySimilarities

Este procedure recebe como paramteros vários face ids, o id de um person group ou de um large person group, um número máximo de candidatos que vão ser retornados e um nível de confiança.

Escolheu-se um large person group id uma vez que esta lista já continha caras (adicionadas no procedure **AddFaceFromURLToLargePersonGroupPerson**) e já estava treinada.

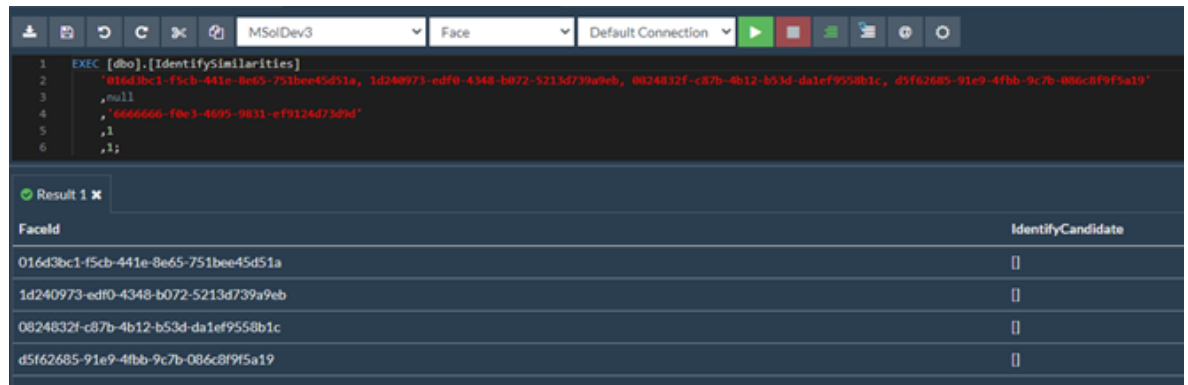


Fig 12. Exemplo de funcionamento do procedure IdentifySimilarities.

10. VerifyFaceToFace

Este procedure recebe dois ids de duas caras detectadas pelo procedure DetectFaceFromURL. Ao detectar duas caras iguais mas com ids diferentes o resultado é o seguinte:

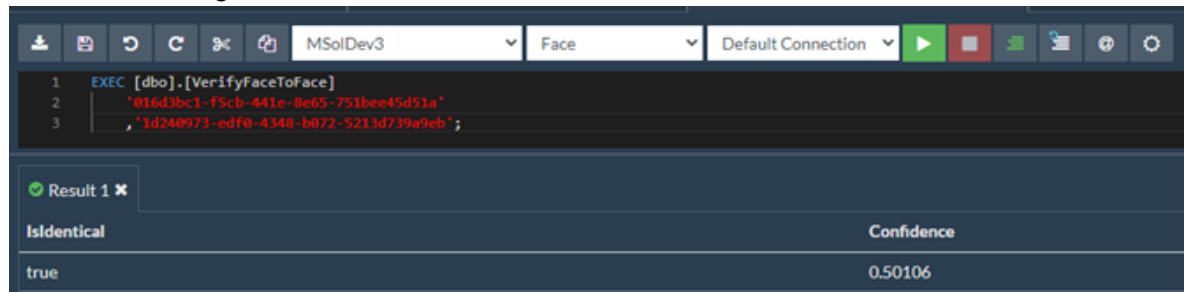


Fig 13. Exemplo de funcionamento do procedure VerifyFaceToFace.

11. VerifyFaceToPerson

Este procedure, ao receber o id referente a uma cara, deteta se essa cara corresponde a uma dada pessoa.

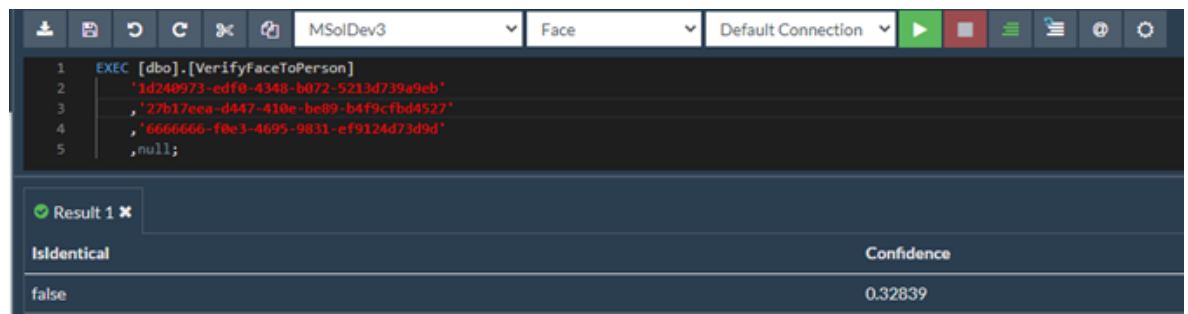


Fig 14. Exemplo de funcionamento do procedure VerifyFaceToPerson.

Exemplos de Funções:

12. ListFacesInLargeFaceList

Este procedure, ao receber o ID de uma dada large face list, retorna todas as

caras presentes na lista.

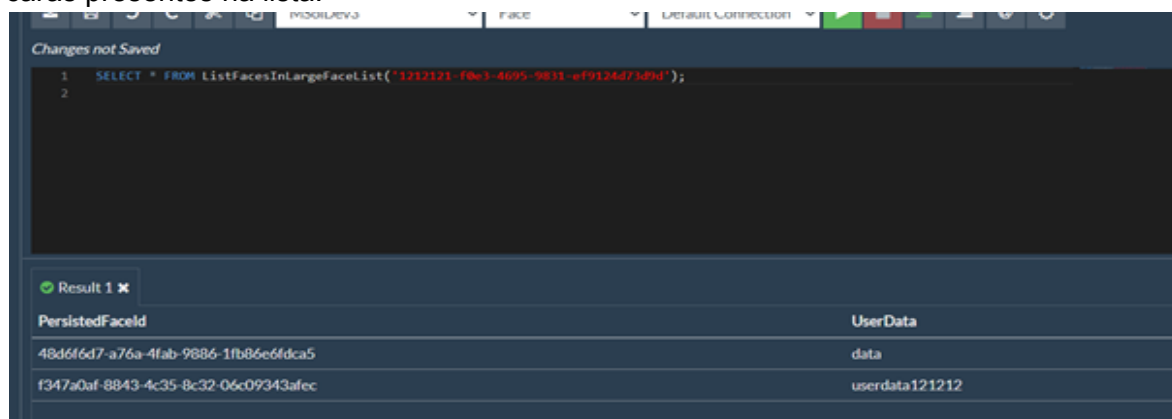


Fig 15. Exemplo de funcionamento da tabela ListFacesInLargeFaceList.

Discussão

O projeto descrito no presente relatório, elaborado com o auxílio da engenheira Filipa e do professor Filipe na Connecting Software, sofreu algumas alterações ao longo de todo o percurso.

O plano inicial seria implementar um conector para a API text analytics, realizar testes unitários, testes de integração, documentação e finalizar com uma aplicação para a Microsoft Teams. Após alguns percalços, o plano sofreu algumas modificações. O objetivo passou a ser a implementação de um conector para a Face API, a elaboração de testes unitários, testes de integração, documentação e a finalização com uma aplicação para a Microsoft Teams. Ao longo do semestre, o conector demorou mais tempo que o esperado pelo que não foi possível realizar a aplicação e os testes de integração.

Para a elaboração do conector programou-se em C# e utilizou-se tecnologias como o Visual Studio, a Connect Bridge, a plataforma da Azure, o GitHub e o TFS. O mesmo cumpriu com todos os requisitos estipulados e os testes unitários forneceram uma maior segurança de que o mesmo estava bem desenvolvido.

Apesar de não ter sido possível continuar com o projeto, os próximos passos seriam a elaboração de testes de integração e, em seguida, a elaboração da aplicação para a Microsoft Teams.

Com isto dito, e assim como esperado, a empresa forneceu um projeto desafiante e com muito conhecimento por adquirir. Não só obtive mais conhecimentos em C# como em bases de dados e em plataformas, como o GitHub, que são frequentemente utilizadas no mundo do trabalho.

Referências

- [1]
<https://www.mxm.com.br/blog/integracao-de-sistemas-o-que-e-e-como-funciona-em-tenda/>
- [2]
https://en.wikipedia.org/wiki/API#History_of_the_term
- [3]
<https://blog.postman.com/intro-to-apis-what-is-an-api/>
- [4]
<https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>
- [5]
<https://azure.microsoft.com/pt-pt/services/cognitive-services/face/>
- [6]
<https://www.m2sys.com/blog/guest-blog-posts/evolution-of-facial-recognition-technology/>
- [7]
<https://www.itpro.com/security/privacy/356882/the-pros-and-cons-of-facial-recognition-technology>
- [8]
<https://docs.microsoft.com/en-us/rest/api/faceapi/face/detectwithstream>
- [9]
<https://westus.dev.cognitive.microsoft.com/docs/services/563879b61984550e40cbe8d/operations/5a157b68d2de3616c086f2cc>
- [10]
<https://docs.microsoft.com/pt-pt/azure/connectors/apis-list>
- [11]
<https://digituma.uma.pt/handle/10400.13/2221>
- [12]
<https://digituma.uma.pt/bitstream/10400.13/2228/1/MestradoF%C3%A1bio%20Caires%20Lu%C3%ADs.pdf>

[13]
<https://ifttt.com/>

[14]
<https://docs.microsoft.com/en-us/microsoftteams/platform/overview>

Anexo 1 - CB Cognitive Services Research

Anexo 2 - CB Cognitive Services Connector for Face Recognition QuickStart Guide [CBMS]

Anexo 3 - CB Cognitive Services Connector for Face Recognition Configuration Guide [CBMS]