INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Bots - Assistente Pessoal em Contexto Cooperativo

RUI MANUEL FERRAZ MOREIRA DA SILVA
Outubro de 2018

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

BOTS – Assistente Pessoal em Contexto Cooperativo

Rui Manuel Ferraz Moreira da Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Constantino Martins

Porto, outubro 2018

Dedicatória

Dedico o trabalho apresentado neste documento à minha esposa Joana Silva, aos meus pais e irmã, e àqueles que sempre me apoiaram na procura e conquista dos meus objetivos e projetos pessoais.

Resumo

Numa altura em que a tecnologia está a evoluir num sentido que permite aumentar a eficiência no trabalho, existe uma crescente tendência de aumentar o número de tarefas. No entanto, continua a ser preciso reportar tudo o que é feito, tanto para efeitos de controlo de tempo necessário para cada tarefa e de gestão de recursos como também, a um nível mais macro, o desempenho dos colaboradores e, por último, da organização.

É necessário manter o nível de detalhe ao reportar tarefas e a precisão dos tempos tomados para a conclusão das mesmas , no entanto, torna-se difícil manter os padrões de qualidade a que a organização se compromete quando o número de tarefas a executar aumenta ao longo do período laboral.

Ao longo deste documento é explorada uma forma rápida e proativa de controlar o registo de horas usando assistentes pessoais virtuais – também conhecidos como *bots* – num contexto cooperativo. Após analisar a opção de fazer uma aplicação ou website chega-se à conclusão de que, apesar da simplicidade na forma de resolver o problema, existem limitações que são resolvidas com a solução exposta neste documento ao facilitar o acesso às funcionalidades principais de forma integrada com canais de comunicação que já são usados frequentemente pelos colaboradores usando linguagem natural. Para reconhecimento da linguagem natural é usado *machine learning* através do serviço LUIS da Microsoft. Este tem a função de reconhecer a intenção do utilizador para que a solução aqui proposta possa agir em concordância.

Para garantir a eficácia e fiabilidade da utilização de um assistente pessoal são realizados testes A/B. Um grupo de controlo usa o método atualmente em vigor, o portal do *Costumer Relationship Manager* (CRM), e outro grupo usa apenas o assistente pessoal virtual como único meio para a realização do registo de horas.

O objetivo é introduzir uma forma mais rápida de introdução de *time reports* no sistema atual da organização sem descurar a eficácia do método atual, o que permite à empresa beneficiar do uso de novas tecnologias e contribuir para o enriquecimento do conhecimento na área.

Com a utilização da solução realizada foi possível acelerar o processo do registo e desacoplar o método usado ao permitir que agora o registo tomasse lugar em plataformas mais *mobile friendly* como Microsoft Teams, Skype e Facebook Messenger.

A principal conclusão que se pode retirar desta experiência é que é relativamente rápido e fácil desenvolver um *bot* minimamente capaz de realizar tarefas simples. No entanto, quando se pretende evoluir para um conjunto mais alargado de parâmetros, tais como intenções complexas envolvendo vários cenários possíveis consoante o contexto, o trabalho exigido para o desenvolvimento de novas funcionalidades aumenta exponencialmente. Segue a regra que tem existido predominantemente nas apps mobile: preferível realizar poucas funções de forma eficaz ao invés de muitas tarefas de forma ineficaz.

Palavras-chave: Assistente pessoal, *Bot*, registo de horas, linguagem natural, *Machine Learning*

Abstract

At a time when technology is evolving in a way that increases work efficiency, there is an increasing tendency to increase the number of tasks. However, it is still necessary to report everything that is done for the purposes of time control of each task, resource management and, at a more macro level, the performance of the employees and, ultimately, the organization.

It is necessary to maintain the level of detail when reporting tasks and the accuracy of the time taken to complete them, and it becomes difficult to maintain the quality standards that the organization undertakes when the number of tasks increases throughout the work period.

Throughout this paper we explore a fast and proactive way to track the recording of hours using virtual personal assistants - also known as bots - in a cooperative context. After analyzing the option to make an application or website, it is concluded that, despite the simplicity in solving the problem, there are limitations that are solved with the solution presented on this document by facilitating access to the main features in an integrated way with channels that are often used by collaborators using natural language. For natural language recognition, it is used machine learning with resource to the Microsoft LUIS service. This will have the function of recognizing the intention of the user so that the solution proposed here can act in agreement.

A / B tests are performed to ensure the effectiveness and reliability of using a personal assistant. One control group uses the currently in force method, the Costumer Relationship Manager (CRM) portal, and the other group will use the virtual personal assistant as the only way to manage time reports.

The goal is to build a faster way of introducing time reports into the organization's current system without neglecting the effectiveness of the current method. It allows the company to benefit from the use of new technologies and contribute to the enrichment of the knowledge in the area.

With the use of this solution, it was possible to accelerate the registration process and decouple the method used by allowing registration to take place on more mobile platforms such as Microsoft Teams, Skype and Facebook Messenger.

The main conclusion that can be drawn from this experience is that it is relatively quick and easy to develop a minimally capable bot to perform simple tasks. However, when one intends to evolve to a wider set of parameters such as complex intentions involving various scenarios

depending on the context, the work required for the development of new functionalities increases exponentially. It follows the rule that has existed predominantly in mobile apps: it is preferable to perform few functions effectively rather than many tasks inefficiently.

Keywords: Personal Assistant, Bots, time reports, natural language, Machine learning

Agradecimentos

Gostaria de agradecer ao meu supervisor Eng. João Sousa por me dar a oportunidade de explorar novas soluções, ajudar no fornecimento de material necessário que me permitiu realizar o estudo na solução presente nesta tese e pelas orientações valiosas durante o desenvolvimento da mesma.

Agradeço também ao meu orientador Prof. Constantino Martins pelas revisões ao meu trabalho e comentários que me permitiram melhorar de forma substancial. Sempre se destacou pelo interesse no tema e vontade de aprender mais sobre o mesmo ao longo deste percurso de estudo e desenvolvimento.

Foram também essenciais para o meu desenvolvimento como engenheiro de software muitas das aulas que frequentei no mestrado de Engenharia de Software no ISEP onde pude explorar diferentes formas de resolver problemas de engenharia comprovadas pela experiência e dedicação dos professores. Destaco disciplinas como Integração de Sistemas lecionado pelos professores Nuno Silva e Paulo Maio, onde aprendi todo um conjunto de técnicas para ligar vários serviços de forma flexível e escalável, Técnicas Avançadas de Software lecionado pelos professores Nuno Malheiro e Isabel Azevedo, onde aprendi a importância de manipular linguagem funcional e usar programação orientada a aspetos (AOP) e Qualidade de Software lecionado pelo professor Alberto Sampaio, onde foi possível aprender um conjunto de formas de testar e avaliar a qualidade de soluções de software.

Por último, agradeço à minha mulher, Joana, pela cooperação e compreensão durante os meses de investigação nos quais participou com motivação e apoio considerável.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

AOL – *America Online*

API – *Application Programming Interface*

CRM – *Customer Relationship Management*

CTA – *Call-To-Action*

DOS – *Denial of Service*

ERP – Enterprise Resource Planning

HTTP – *Hypertext Transfer Protocol*

AI – Artificial Intelligence

ICMP – *Internet Control Message Protocol*

IRC – *Internet Relay Chat*

LUIS – *Language Understanding Intelligent Service*

SDK – *Software Development Kit*

SLA – *Service Level Agreement*

SSL – *Secure Sockets Layer*

1 Introdução

Neste capítulo irá ser explorado o problema de haver tarefas repetitivas, no contexto do dia de trabalho de um colaborador de uma organização, e que podem ser analisadas pela sua natureza composta por padrões previsíveis. Após a demonstração de um cenário real, é feita uma breve análise das opções para tentar ajudar no processo de registo de horas de um colaborador. É dada como proposta de resolução a elaboração de um assistente pessoal virtual capaz de acelerar o processo de registo de horas, sendo também capaz de manter o contacto com o mesmo de uma forma semelhante a uma conversa natural com um assistente. (Thomas, 2016)

1.1 Interpretação do problema

As organizações têm um desafio constante com a gestão de tarefas pois envolve frequentemente a capacidade de organização e motivação por parte dos colaboradores.

A falta de organização do colaborador impede de haver registo pormenorizado das tarefas realizadas de forma regular. Após diversos relatos de incumprimento de registos de horas por parte dos gestores de projeto, foi necessário averiguar formas de tentar minimizar este comportamento. Um dos problemas mais evidentes apontado pelos colaboradores deve-se ao método atual ser pouco prático. Consequentemente, estar repetitivamente a passar por essa má experiência tem como consequência a perda de motivação por parte dos colaboradores. Como resultado dessa desmotivação, os colaboradores sentem-se mais relutantes a aceitar novos desafios e tarefas variadas por pensar na dificuldade acrescida que advém disso.

Atualmente existem formas de motivar os colaboradores que envolvem atribuir tarefas que normalmente seriam realizadas por colegas de nível mais elevado de responsabilidades. A esta

prática dá-se o nome de *Job Enrichment*. Desta forma promove-se o sentimento de que o colaborador está a executar tarefas que fazem uma maior diferença na organização. (Umstot, 1976)

No entanto, como resultado desta prática, existe um grande número de tarefas repetitivas que requerem um empenho significativo de tempo para as realizar. Atualmente, e cada vez mais, o tempo é precioso e como tal, é natural que o desejo por automatizar algumas dessas tarefas seja favorável para o colaborador.

A filosofia de utilização de um computador para realizar tarefas repetitivas levou à procura de formas para tornar um processo lógico num processo automatizado e/ou simplificado de forma a deixar mais tempo e atenção para o ser humano dedicar a outras tarefas mais pertinentes. Isto revelou-se extremamente importante para garantir uma maior produtividade e otimização de recursos. (Kongthon et al, 2009)

Tal como se verificou durante a revolução industrial, entre outros casos, sejam pequenas ou grandes tarefas, as organizações procuram sempre novas formas de automatizar os seus processos. Por forma a haver controlo numa fase inicial de transição, seria ideal que essas mesmas tarefas fossem supervisionadas com recurso a uma interação humana simplificada, mas sem descurar da necessidade de tomada de decisão do colaborador como forma de acompanhamento.

Neste caso específico, a motivação para explorar novas formas de automatizar o processo de registo de horas advém da atual solução não ir ao encontro da satisfação do colaborador.

A solução atual é integrada com o produto de *Customer Relationship Management* (CRM) da Microsoft com um *plugin* desenvolvido internamente com as entidades necessárias para a lógica de negócio inerente ao registo das horas, gestão de projetos e respetivos *budgets*.

Descrito neste documento como CRM, está representado como Microsoft Dynamics 365, um produto da Microsoft que unifica capacidades de CRM com ERP (*Enterprise Resource Planning*) em aplicações que funcionam perfeitamente em conjunto através de vendas, atendimento ao cliente, serviço no campo, operações, finanças, marketing e automatização de serviços de projeto. (Dynamics 365, 2017)

Atualmente, o colaborador precisa de aceder ao CRM e dar início a um processo que não segue boas práticas de *User Experience* por ser consideravelmente lento e complexo originando numa fraca motivação para se lembrar de o fazer todos os dias. Como conclusão, é recorrente que

haja cenários em que só após vários dias é que o colaborador sente a necessidade de atualizar a situação do registo de horas. Nesta fase, a memória do mesmo já não é tão clara e o detalhe do seu registo perde-se consideravelmente.

Para o efeito pode-se explorar a hipótese de desenvolver uma plataforma onde seja possível o colaborador criar fluxos de ação para tarefas rotineiras e agir sobre esses fluxos quando quisesse. Teria de haver suporte a tarefas agendadas e lembretes associados. Tal plataforma seria relativamente complexa pois envolve construir toda uma interface de interação que possa funcionar numa comunicação bilateral.

As opções a considerar compreendem uma plataforma *web* acessível a partir de um *browser*, uma aplicação móvel e um assistente pessoal virtual.

Após a análise que se segue neste documento chega-se à conclusão de que o desenvolvimento de um assistente pessoal virtual é a opção que mais se encaixa nas necessidades da organização.

Um assistente pessoal virtual terá de ser capaz não só de informar e aconselhar as ações indicadas ao utilizador, mas também realizar ações em nome do mesmo. No entanto, ao longo deste documento, o conceito de assistente pessoal virtual terá como sinónimo a designação de *bot*, visto que este é o termo correto para o contexto, do elemento que contém a lógica necessária para executar as ações propostas à realização das tarefas.

É então necessário haver alguma forma de delegar tal género de tarefas de forma a não consumir mais tempo que o estritamente necessário ao colaborador e que ainda seja capaz de minimizar a probabilidade de se adiar ou acumular a referida tarefa.

É de evitar este último comportamento pois existe uma extrema importância para as métricas de desempenho que o colaborador terá de apresentar à organização. Num nível mais macro, a própria organização terá de ter esta componente bem alinhada com as normas de qualidade a que se compromete. No entanto, não é habitualmente contabilizado o tempo e esforço que o colaborador dedica neste registo, por vezes, exaustivo e detalhado para corresponder com os padrões de qualidade.

O colaborador pode ainda optar por realizar esta tarefa fora do horário laboral. No entanto, esta prática compromete o equilíbrio entre a vida profissional e pessoal.

Para esse efeito é demonstrado um *use case* (Figura 1 – Interação de um *bot* que agenda reuniões com o utilizador) que ajuda a compreender o que pode ser esperado de um *bot*.

Quando se pretende automatizar um processo de registo de horas é necessário criar um algoritmo responsável por:

- Identificar o colaborador;
- Verificar se para o dia atual contém as 8 horas laborais registadas;
- Caso falte horas para registar deve sugerir o registo para a última tarefa registada;
- Caso o utilizador indique que pretende novo registo deve começar o formulário para novo registo;
- Pedir ao colaborador o nome do projeto para fazer uma pesquisa e apresentar um conjunto de opções para escolher;
- Obter a lista de *budgets* para o projeto selecionado para o utilizador poder escolher;
- Sugerir quantas horas deve registar consoante as restantes para o dia em questão;
- Permitir o colaborador introduzir o número de horas que pretender;
- Permitir o colaborador introduzir o número real de horas que trabalhou para o presente projeto e *budget*;
- Permitir o colaborador introduzir detalhes sobre o que foi realizado no decorrer da tarefa que está a registar;
- Mostrar um resumo das informações recolhidas para haver confirmação por parte do colaborador;
- Caso o colaborador queira retificar alguma informação terá de percorrer os passos anteriores com a adição da opção de poder passar à frente a informação que considera correta;
- Fazer o registo e mostrar mensagem de confirmação ao colaborador.

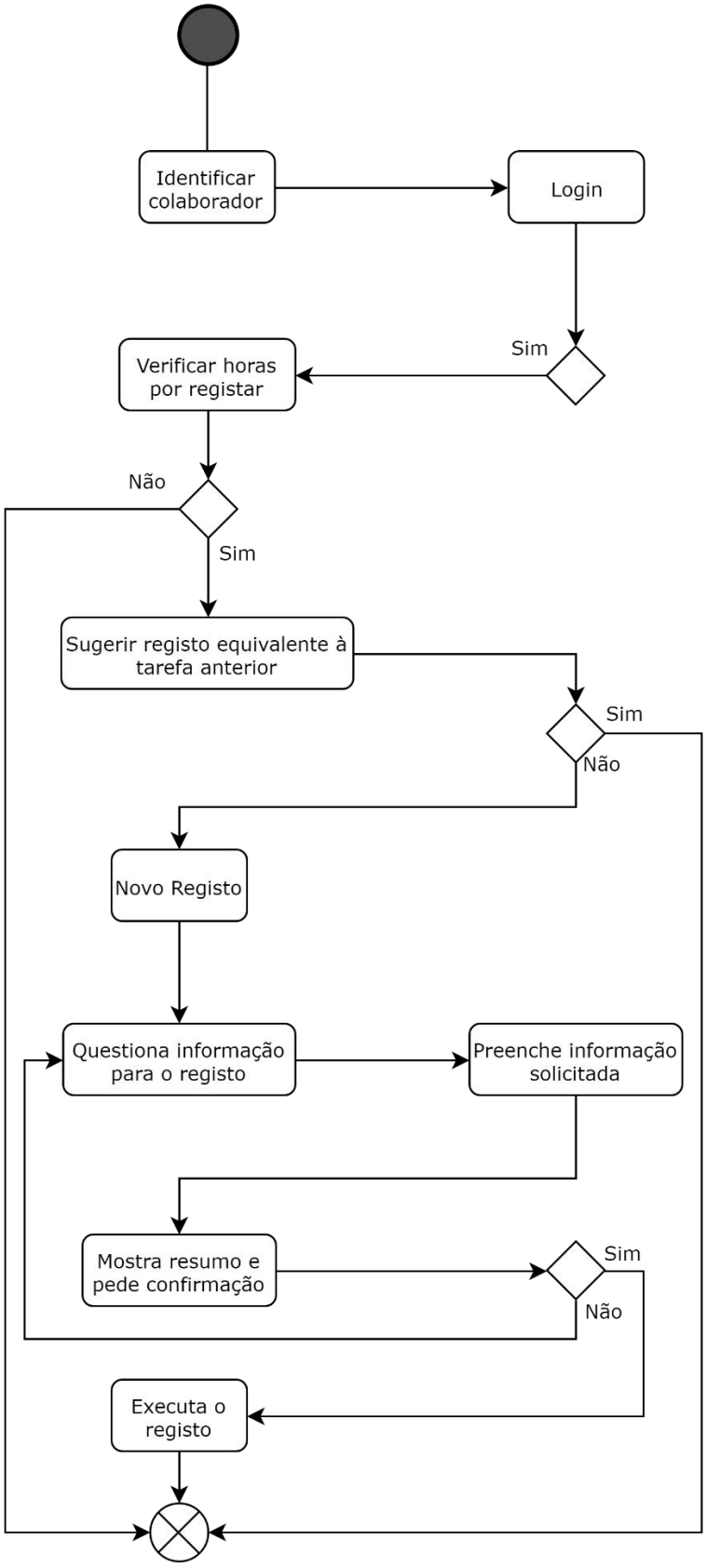

Figura 1 – Interação de um *bot* que agenda reuniões com o utilizador

1.2 Objetivos

O objetivo geral deste trabalho é a construção e implementação de um assistente pessoal que dê suporte à gestão do registo de horas do colaborador. No caso da solução que se procura, o equivalente representa-se como um assistente pessoal virtual que é capaz de entender intenções especificas do utilizador. Essas intenções representam ações que o utilizador delega ao assistente pessoal.

Podem ser usados meios de comunicação não convencionais como a conversação com computador, que facilite a realização de tarefas repetitivas através de padrões analisados e avaliar em que medida a solução implementada é eficaz na resolução do problema a que se propõe, sem comprometer a eficácia comparando com os métodos a que se propõe substituir e/ou complementar.

Os objetivos específicos deste trabalho que permite concretizar o objetivo geral são:

- Criar um subtipo de *bots* caracterizados como *personal-assistant* (assistente pessoal) para permitir haver um acompanhamento do registo de horas de cada colaborador e de forma proativa o lembrar dessa tarefa, facilitando até a sua inserção ao fazê-lo pelo mesmo meio de comunicação apenas demonstrando essa intenção ao *bot*, sem ter de visitar o portal *web* do CRM para o mesmo efeito;
- Evitar a necessidade de haver instalação de *software* adicional para realizar a tarefa evitando também a preocupação por parte dos utilizadores mais conscientes desta área que é cada vez mais relevante nos dias correntes;
- Ainda em relação à segurança, por este tipo de solução apenas depender de um meio de comunicação que sabe quem é o utilizador, podem ser usados canais de comunicação como SMS a partir do serviço *Twilio* (depende dos dados que se consegue obter sobre o colaborador como o número de trabalho ou pessoal) ou outros meios como o email, Skype, Facebook Messenger (depende da informação necessária para poder autenticar o utilizador no CRM da organização), Microsoft Teams e Cortana.

Poderão ser explorados métodos de recomendação de ações a seguir baseando em técnicas de *machine learning* que dará a capacidade de o assistente pessoal sugerir opções de repetição de atividades anteriormente relevantes de forma a otimizar a sua utilização.

1.3 Motivação

Por ser uma organização com fortes ligações à área da informática, é identificada a solução ao usar métodos automáticos de realizar o registo de horas. Para tal, são identificados os padrões necessários para realizar a tarefa.

Robots, também conhecidos como *bots,* são reconhecidos por realizar tarefas consoante um conjunto de padrões e regras repetitivas, portanto a ideia passa pelo desenvolvimento de um *bot* que apresente uma solução para o problema.

Foi proposto o desenvolvimento de um *bot* com o objetivo de automatizar o registo de horas com base num número de informações que podem ser fornecidas de forma mais rápida e cómoda pelos colaboradores, podendo haver integrações com vários canais de comunicação compatíveis com meios mais portáteis como o telemóvel.

O conceito de utilizar um *bot* deve-se ao crescente interesse que gerou nos últimos 5 anos como se pode confirmar pela quantidade de pesquisas feitas no Google (Figura 2 – Tendência de pesquisas no Google pela palavra "*chatbot*" nos últimos 5 anos).

Figura 2 – Tendência de pesquisas no Google pela palavra "*chatbot*" nos últimos 5 anos (Chatbot search trend, 2017)

O termo *chatbot* é uma variação representativa de assistentes pessoais virtuais.

1.3.1 Identificação da oportunidade

Os funcionários da organização são confrontados diariamente com a necessidade de registar as horas de trabalho em diferentes contextos. Este trabalho não é pertinente para a realização das tarefas, mas é preciso ser feito para haver controlo de gastos/recursos. O número de passos e

a inflexibilidade para registar as horas é desgastante e consome muito tempo, sendo necessário abrir o portal do CRM.

1.3.2 Análise de oportunidade

A organização identificou a necessidade de automatizar a tarefa de registo de horas porque existem pessoas que têm tendência a esquecer ou a não ter tempo para o fazer.

Havendo possibilidade de introduzir novos métodos de trabalho que ajudem a poupar tempo e a tornar o processo de interação mais interessante, resultando numa tendência de aumento da satisfação dos colaboradores.

Foi feito um questionário (ver Anexo 1 – Questionário de satisfação do Portal CRM) onde é analisado o grau de satisfação atual dos colaboradores com a utilização do portal de CRM.

Com um total de 73 respostas, os resultados na Figura 3 – Valores médios de satisfação comprovam que existe a oportunidade de investir numa pesquisa e consequente desenvolvimento, pois existe também uma componente que impacta a precisão dos registos com os atuais métodos que podem ser melhorados e refletirem numa melhor gestão económica da organização.

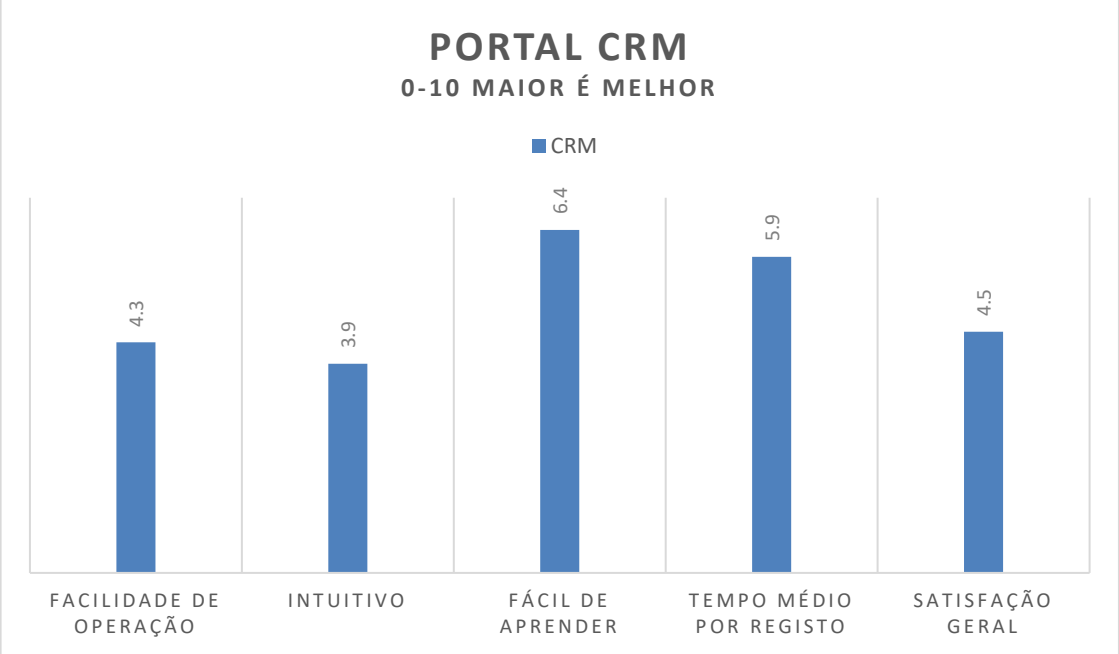

Figura 3 – Valores médios de satisfação da utilização do portal de CRM

Foi então sugerida a criação de uma aplicação para dispositivos móveis que fizesse o mesmo trabalho, mas continuava a necessidade de instalar e preparar o ambiente onde o colaborador pode fazer o registo de horas. Outro problema com as aplicações é que uma considerável parte das pessoas tem receio do nível de permissões que é necessário conceder à mesma para cumprir a sua função. Alguns utilizadores questionam se as respetivas aplicações realmente precisam de certas permissões que outrora podem pôr informação sensível em risco e/ou sentirem a sua privacidade comprometida. (Finjan Mobile, 9)

Ao usar um assistente pessoal virtual pode-se recorrer a canais de comunicação mais flexíveis e com integração a aplicações sociais mais conhecidas por parte dos colaboradores. Tais aplicações podem ser o Skype, Microsoft Teams e Facebook Messenger. A preocupação levantada pelas permissões exigidas para a instalação de uma aplicação própria já não se põe em causa.

1.4 Contributos

Ao desenvolver este trabalho houve oportunidade de explorar uma área da informática que tem cada vez mais relevância nos dias correntes. Engloba uma temática muitas vezes considerada polémica quando se trata de criar e usar a Inteligência Artificial (AI) para realizar tarefas outrora realizadas pelos humanos. Neste trabalho, esta vertente é usada essencialmente para reconhecer as intenções passadas pelos colaboradores sob forma de conversa corrente sendo que após obtermos esse resultado, todas as restantes tomadas de decisão do *bot* são realizadas por codificação estática de condições predefinidas e não com base em esquemas de probabilidades comumente relacionadas com o trabalho realizado pela AI.

É, no entanto, de realçar que na componente que envolve AI para reconhecer as intenções tem a particularidade de melhorar o reconhecimento, fazendo com que quanto mais interação tiver, maior é a probabilidade de obter resultados mais fidedignos com o que o colaborador pretende realmente alcançar. Esta contribuição é importante para a eficácia e consequente adesão à utilização da solução.

O meio de comunicação é outra componente presente nesta solução que difere do que normalmente se vê nas soluções de software, usando como meio de introdução de dados (input) texto corrente ao invés de apenas botões que requerem uma navegação e fluxo de trabalho

predeterminado. O texto corrente como forma de comunicação entre colaborador e o *bot* pode ser usado sob a sua forma mais tradicional usando a escrita por computador com recurso ao teclado.

> "Fast and Reliable: Every app is designed with a visual interface like buttons, texts, images and much more. what if there is system which uses language as an interface, it is not difficult for us as language is taught for us since we are born. similarly chat-bots uses language as an interface to communicate like a normal human being thus making it easy to understand and use than any other technology ever created." (Chatbot vs Traditional Service Apps, 2017)

Tendo um serviço capaz de converter texto em intenções com reconhecimento de parâmetros – tais como a data, horas e projeto – o passo seguinte será fazer com que esse texto possa chegar através de fala, tendo primeiro de passar por um processo de conversão para texto.

Juntando estes componentes e serviços no desenvolvimento de um assistente pessoal virtual permite explorar todo um novo paradigma de interação entre o ser humano e as máquinas.

1.4.1 Contributos ao nível empresarial

Na maioria das equipas da empresa, todos os elementos têm de ser capazes de fazer desenvolvimento em qualquer projeto/produto/cliente sem grande período de adaptação. Há então a necessidade de poder registar os *time reports* de cada produto com precisão. O desenvolvimento de um assistente pessoal virtual é capaz de trazer o contributo de ajudar a acelerar este processo com um conjunto de pequenas frases capazes de indicar a mudança de foco.

O contributo deste trabalho estende-se particularmente bem aos gestores, visto que têm uma visão mais atualizada do estado dos *time reports* dos seus profissionais. Revela-se importante para obter uma visão financeira dos projetos.

1.4.2 Outros contributos

Como contributos para a comunidade espera-se melhorar o reportório e experiência na área.

A empresa realiza periodicamente, como forma de fornecer formação aos seus colaboradores, sessões internas de passagem de conhecimento denominadas como "Conhecimento Sobre Investigação (CSI)" onde os participantes ficam a conhecer as metodologias e tendências atuais

do mercado. Após essas sessões é partilhada documentação sobre o tema com todos os envolvidos.

1.5 Organização da Dissertação

Este trabalho está estruturado em 8 capítulos organizados da seguinte forma:

- No Capítulo 1 – Interpretação do problema, pretende-se expor o problema identificado e apresentar uma proposta de solução para o mesmo, ilustrando-o com exemplos. São também apresentados os objetivos gerais e específicos deste trabalho assim como os seus contributos a nível empresarial e outros;
- No Capítulo 2 – É apresentado o estado da arte apresentando um resumo do que é descrito na literatura e as soluções postas em prática fazendo inclusive uma comparação breve entre elas. Também é apresentada uma visão histórica e a análise de valor;
- No Capítulo 3 – Design da solução, descreve-se a solução proposta neste trabalho e a sua estrutura de forma mais detalhada;
- No Capítulo 4 – Construção da solução, é clarificada a forma de implementação da solução;
- No Capítulo 5 – Avaliação da solução, enuncia-se uma crítica/apreciação da solução implementada e dos resultados obtidos;
- No Capítulo 6 – Conclusão sobre a experiência obtida na realização deste trabalho e possíveis oportunidades de melhoria;
- No Capítulo 7 – É apresentada uma listagem de referências utilizadas para orientação e estrutura deste trabalho;
- No Capítulo 8 – Por fim, são apresentados os anexos usados como argumentação adicional ao trabalho.

2 Estado da Arte

Neste capítulo vai ser retratado o estado atual da tecnologia que é usada para o desenvolvimento da solução proposta, começando por definir o que é um *bot* e o seu contexto na realização de um assistente pessoal.

2.1 Conceito e definição de *Bot*

Um *bot*, também conhecido como *robot* ou *web robot*, é um programa computorizado com o intuito de realizar ações humanas que seguem certos padrões de forma repetitiva.

Quando um *bot* é controlado através da interação com um utilizador que diz o que deve ser feito então tem-se o conceito de um assistente pessoal virtual.

Na solução proposta para este documento, o termo assistente pessoal virtual representa o método usado para a comunicação entre o colaborador e o *bot* propriamente dito.

O termo *bot* é usado para descrever a perspetiva da lógica envolvida nos processos e a comunicação entre serviços.

2.2 Visão histórica do *Bot*

Os primeiros *bots* a surgirem na internet foram em 1988 quando surgiu o IRC (*Internet Relay Chat*). Tinham como nome *Jyrki Alakuijala's Puppe*, *Greg Lindahl's Game Manager* (para o jogo Hunt the Wumpus) e *Bill Wisner's Bartender*. Para os servidores onde estavam hospedados não fecharem devido ao tempo de inatividade, estes *bots* forneciam diversos serviços automatizados aos utilizadores a partir de um canal de comunicação.

Outros *bots* que foram surgindo desde os primórdios da Internet foram os *crawlers* da *web* para os primeiros motores de busca. O primeiro *bot* usado para indexar páginas da *web* foi o *WebCrawler*, criado em 1994. Foi usado pela AOL (*America Online*) em 1995 e depois comprado pela *Excite* em 1997. O mais famoso rastreador da internet, o *Googlebot*, foi originalmente chamado *BackRub* quando foi criado em 1996.

Existem também os *bots Sub7* e *Pretty Park* que ficaram na história como os primeiros *bots* destinados a ações maliciosas, conhecidos como *botnets*, sendo categorizados como cavalo de troia e um *Worm*, respetivamente, quando foram publicados no IRC em 1999. O objetivo destes *bots* era instalar-se secretamente em máquinas que se conectam a um canal IRC para que eles pudessem ouvir comandos mal-intencionados.

O próximo programa *botnet* que é preciso reconhecer é o *GTbot*, publicado no IRC em 2000. Este *bot* era um programa falso, que se mascarava do programa de cliente *mIRC*, capaz de lançar ataques DOS (*Denial of Service* ou ataques de negação de serviço). Nos anos que se seguiram, *hackers* foram criando outros *botnet* que eram capazes de usar máquinas infetadas para realizar vários ataques, como *ransomware* e roubo de informações. Ao longo do tempo, os *botnets* deixaram de usar o IRC como forma de comunicação, passando a usar HTTP (*Hypertext Transfer Protocol*), SSL (*Secure Sockets Layer*) e ICMP (*Internet Control Message Protocol*) como outros meios de se propagarem pela internet, há medida que esta foi crescendo e evoluindo.

A prevalência de *botnets* continuou a crescer e é considerada por especialistas como a ferramenta favorita dos *hackers*. Um dos maiores *botnets*, emergentes em 2007, foi chamado de "*Storm*". Estima-se que este *bot* tenha infetado até 50 milhões de computadores e foi utilizado para muitos tipos de crimes, incluindo a fraude nos preços das ações e roubo de identidade.

Os *botnets* desempenharam um papel importante na epidemia de spam através do email. Em 2009, um programa de *botnet* chamado *Cutwail* foi usado para enviar 74 mil milhões de emails de *spam* por dia.

Apesar da fama dos *bots* serem reconhecidos pelos fins eticamente questionáveis, operam sempre sobre a mesma premissa: tarefas repetitivas e automatizáveis. (Knecht, 2016)

2.3 Visão histórica do Assistente Pessoal virtual

Um assistente pessoal virtual, também chamado de assistente com inteligência artificial ou assistente digital, é um programa de computador que entende comandos de texto e voz em linguagem natural e conclui tarefas para o utilizador.

Tais tarefas, historicamente desempenhadas por um assistente pessoal ou secretário, incluem tomar notas, ler texto ou mensagens de e-mail em voz alta, procurar números de telefone, fazer chamadas e lembrar o utilizador sobre compromissos agendados. Atualmente, os assistentes

virtuais populares incluem o Amazon Alexa, a Siri da Apple, o Google Now e o Cortana da Microsoft - o assistente digital integrado ao Windows 10.

2.3.1 Tipos de assistentes virtuais

Embora a definição se concentre na forma digital de assistentes pessoais, o termo assistente pessoal virtual também é comumente usado para descrever colaboradores contratados que trabalham em casa realizando tarefas administrativas normalmente executadas por assistentes executivos ou secretários.

Os assistentes virtuais também podem ser comparados com outro tipo de programação de inteligência artificial voltada para o consumidor, chamados de consultores inteligentes.

2.3.2 Tecnologia

Os assistentes pessoais virtuais são geralmente programas baseados em nuvem que exigem que os dispositivos e/ou aplicações estejam ligados à internet para que funcionem. Três dessas aplicações são a Siri em dispositivos Apple, Cortana em dispositivos Microsoft e Google Assistant em dispositivos Android.

Também existem dispositivos dedicados a fornecer a experiência de um assistente pessoal virtual. Os mais populares estão disponíveis na Amazon, Google e Microsoft. Para usar o assistente virtual do Amazon Echo, chamado Alexa, os utilizadores chamam pela palavra de ativação "Alexa". Uma luz no dispositivo sinaliza ao utilizador que está pronto para receber um comando, que normalmente envolve solicitações de linguagem simples, como "qual é o tempo para hoje" ou "reproduzir música pop". Essas solicitações são processadas e armazenadas na nuvem da Amazon.

As tecnologias que dão poder a assistentes pessoais virtuais exigem enormes quantidades de dados, que servem para alimentar plataformas de inteligência artificial, incluindo *machine learning*, processamento de linguagem natural e plataformas de reconhecimento de voz. À medida que o utilizador interage com um assistente pessoal, a programação de inteligência artificial usa algoritmos sofisticados para aprender e melhorar a previsão das necessidades do utilizador.

2.3.3 Capacidades de um assistente pessoal virtual

Os assistentes pessoais virtuais normalmente executam tarefas simples em nome dos utilizadores, como adicionar tarefas a um calendário, fornecer informações que normalmente seriam pesquisadas na web por meio de um *browser* ou controlar e verificar o estado dos dispositivos domésticos inteligentes, incluindo luzes, câmaras e termostatos.

Os utilizadores também solicitam assistentes pessoais virtuais para fazer e receber chamadas telefónicas, criar mensagens de texto, obter direções, receber notícias e informações meteorológicas, encontrar hotéis ou restaurantes, verificar reservas de voos, ouvir música ou jogar.

2.3.4 Preocupações com a privacidade do assistente pessoal virtual

Alguns consumidores expressam preocupações com a privacidade de assistentes pessoais, como o Amazon Alexa e o Google Home, porque esses assistentes virtuais exigem grandes quantidades de dados pessoais e estão sempre a ouvir para responder aos comandos de voz. Os assistentes pessoais virtuais, em seguida, mantêm interações de voz e informações pessoais para melhorar a experiência do usuário.

A Cortana, por exemplo, funciona melhor usando dados do dispositivo do utilizador, incluindo e-mails e outras comunicações, contatos, dados de localização, histórico de pesquisa e dados de outros serviços da Microsoft que os usuários escolhem para se conectar. Os utilizadores podem optar por não partilhar os seus dados com a Cortana e inclusive podem ajustar as permissões para impedir que determinados dados sejam recolhidos, embora essas ações limitem a utilidade do assistente pessoal.

Os fornecedores de assistentes virtuais também mantêm políticas de privacidade, que definem como cada empresa usa e compartilha informações pessoais. Na maioria dos casos, as empresas não compartilham informações de identificação do cliente sem o consentimento de um cliente.

2.3.5 O futuro dos assistentes pessoais virtuais

Os assistentes pessoais estão a evoluir rapidamente para fornecer mais recursos e valor aos utilizadores. À medida que o reconhecimento de fala e o processamento de linguagem natural avançam, também aumentará a capacidade de um assistente virtual de entender e executar comandos mais variados. E, à medida que a tecnologia de reconhecimento de voz melhora, o

uso de assistentes pessoais virtuais vai se aprofundar nos fluxos de trabalho de negócios (Rouse & Botelho, 2017).

As empresas principais pelo desenvolvimento e evolução desta tecnologia procura também democratizar a mesma para que exista mais liberdade para quem procura desenvolver assistentes pessoais virtuais para domínios de negócio específicos.

2.4 Análise de valor

No resultado da investigação no desenvolvimento desta solução foram encontrados diversos valores acrescentados que influenciam diretamente o colaborador. Este é o utilizador final que interage com a solução e numa primeira abordagem é possível reparar que aumenta a eficiência com que realiza o seu trabalho fazendo com que o registo de horas das suas tarefas seja sob uma forma de interação menos aborrecida através desta nova plataforma mais interativa e comoda em comparação às contantes visitas ao portal de CRM. Com melhoramentos a nível de reconhecimento de padrões nos dias de trabalho de um mesmo colaborador, é possível poupar tempo precioso nos casos em o mesmo realizou a mesma tarefa que o dia anterior precisando apenas indicar essa intenção à assistente pessoal.

O cliente, neste caso a equipa responsável por analisar o desempenho e percurso dos colaboradores da organização, também irá beneficiar com dados de maior precisão, e visto que irá reduzir o tempo despendido pelos colaboradores para esta tarefa, também irá resultar numa maior produtividade. Tendo em conta que os colaboradores irão ser lembrados por uma assistente pessoal proativa para a realização do registo de horas, também o cliente irá ter atualizações mais frequentes deste registo e com mais detalhes, sendo que a probabilidade de um colaborador se lembrar do que fez durante o próprio dia em que está a preencher o registo aumenta drasticamente. Dois outros pontos a concluir destas condições são resultados com maior confiança e maior qualidade dos dados recolhidos.

No entanto, é preciso ter a noção que o tempo, e inerente custo, da investigação no desenvolvimento da solução é um sacrifício que é tomado de forma a se confirmar se compensa o resultado.

2.5 Soluções existentes

Os *bots* podem ser ferramentas essenciais na automatização de tarefas que exigem interação com uma ou mais pessoas envolvidas. Existem *bots* para fazer de tudo e são capazes de entender linguagem natural para compreender o contexto da ação que o utilizador quer que seja realizada.

Existem várias empresas com soluções inteiramente pensadas e desenvolvidas para esta tendência de dotar os computadores que nos rodeiam com Inteligência Artificial usando os *bots* como uma das interfaces de comunicação mais natural para o ser humano transpondo a forma atual de interação a partir de objetos físicos adaptados á realidade da máquina como o teclado e rato.

2.5.1 Amazon

A *Amazon* tem realizado esforços para construir um conjunto de ferramentas com o objetivo de tornar a criação de interfaces conversacionais uma componente de integração para aplicações. Este serviço chama-se *Amazon Lex* e pode ser usado texto e voz para estabelecer comunicação com o *software* [Amazon Lex, 2017]. Como é um serviço baseado na *Cloud* pode facilmente escalar conforme as necessidades. Integra bem com outros serviços fornecidos pela empresa como *Polly* (Amazon) para o reconhecimento automático de áudio para texto e vice-versa. Requer subscrição dos serviços que o desenvolvedor precisar para construir a solução.

2.5.2 Facebook

O *Facebook* usa a popularidade do *Messenger* para divulgar e estabelecer comunicação de *bots* feitos pelos *developers* com os utilizadores da sua rede social [Facebook Messenger's Bot, 2016]. O *Messenger* tem mais de 900 mil utilizadores ativos todos os meses. O *Facebook* dotou a plataforma do *Messenger* com a capacidade de mostrar outras formas de introdução de informação usando CTA (*Call-To-Action*) (Figura 4 – *Messenger Call-To-Action*).

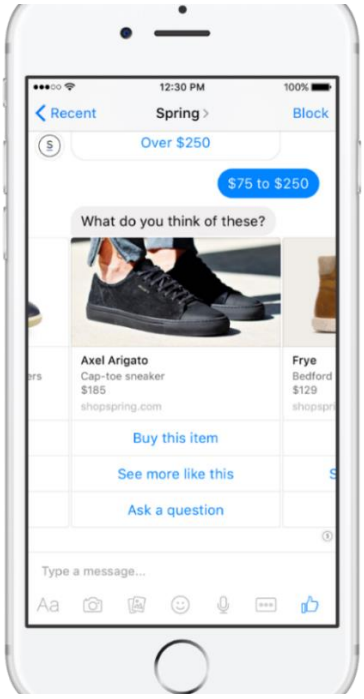

Figura 4 – *Messenger Call-To-Action*

2.5.3 Microsoft

A *Microsoft* tem desenvolvido serviços cognitivos que facilitam a criação de *bots* tais como o serviço LUIS (*Language Understanding Intelligent Service*) para fazer reconhecimento e tradução de linguagem natural para intenções em que o *bot* pode responder.

Esses serviços são consumidos através de APIs (*Application Programming Interfaces*) que possibilitam um conjunto alargado de recursos para o desenvolvimento de *bots* de uma forma transversal, podendo inclusive usar conversão de voz para texto e vice-versa.

A *Microsoft* anunciou a *Bot Framework* para a criação de *bots* usando todas as capacidades das APIs que dispõe para enriquecer o comportamento dos mesmos.

É disponibilizado o SDK (*Software Development Kit) Bot Builder* que permite desenvolver o *bot* usando *NodeJS* e *.Net*. Para testar existe ainda o *Bot Framework Emulator* compatível para *Windows* e *Mac*.

Como meios de comunicação, a *Microsoft* também disponibiliza o *Skype* e *Teams*. No entanto, como um *bot* criado sob *Bot Framework* segue padrões definidos, pode também haver integrações com muitos outros clientes de comunicação como o *Slack*, *Facebook Messenger*, *Telegram*, *GroupMe*, *Twilio*, *Kik* e email.

2.5.4 Outras soluções / abordagens

Para além das opções anteriores, existe uma grande procura e interesse pelo desenvolvimento de ferramentas que ajudem a *developers* a integrarem algum tipo de comunicação semelhante ao dos *bots* no seu *software* usando Python, R e outras linguagens relevantes no mundo do desenvolvimento de IA (Inteligência Artificial) e *Machine Learning*.

Uma grande parte destas ferramentas são disponibilizadas em repositórios públicos no Github e BitBucket.

2.5.5 Botkit

Botkit foi desenvolvido para facilitar o processo de desenvolvimento de *bots* usando conectores a várias plataformas (Brown, 2015). É um projeto *open source* com o objetivo de tornar a ferramenta mais abrangente. Dispõe de um ambiente de desenvolvimento de *bots* chamado *Botkit Studio*.

2.6 Avaliação de soluções e abordagens existentes

São analisadas as diferentes soluções tendo em conta os requisitos funcionais e não funcionais propostos.

São analisadas as soluções apresentadas pela Amazon, Facebook e Microsoft pois são as que prometem melhores resultados devido ao nível elevado de investimento apenas capaz pelas organizações a que pertencem.

Requisitos funcionais:

- Reconhecimento da identidade do utilizador;
- Compreensão do contexto.

Requisitos não funcionais:

- Facilidade de interação;
- Elevado grau de confiança nos resultados.

2.6.1 Requisitos Funcionais

Neste conjunto de requisitos pode-se avaliar como as diferentes soluções se equiparam quando um conjunto de dados de entrada (*input*) é processado, qual o comportamento obtido por cada e o resultado (*output*) que se consegue obter.

2.6.1.1 Reconhecimento da identidade do utilizador

O Facebook permite reconhecer a identidade dos utilizadores através do seu sistema de autenticação próprio que favorece a utilização do Messenger como meio principal de comunicação segura. No entanto, a organização não considera vantajoso considerar este tipo de credenciais profissional para a infraestrutura atual.

A Amazon permite os utilizadores autenticarem-se através de credenciais no AWS (Amazon Web Services) com níveis de permissão para aceder às API do serviço de *chatbot* Amazon Lex que podem ser obtidos através do serviço AWS Identity and Access Management (IAM). No geral é mais complicado, principalmente pela natureza consideravelmente diferente das condições e tecnologias utilizadas pela organização.

A Microsoft permite identificar os utilizadores a partir de vários providers usando o protocolo OAuth 2.0 de autenticação dando uma maior facilidade para contas Microsoft federadas em Azure AD com e sem ADFS. A organização já usa este método de autenticação com todos os colaboradores, portanto pretende escolher este método para autenticar e reconhecer a identidade dos utilizadores que utilizarem esta solução.

2.6.1.2 Compreensão do contexto

Este tópico implica que o assistente pessoal virtual a ser desenvolvido deve ser capaz de reconhecer a intenção que o utilizador quer transmitir usando linguagem natural. Essa linguagem vem sobre várias formas sendo que as mais usadas são: por texto e por voz.

O Facebook contém uma ferramenta nativa para este efeito intitulada como Natural Language Processing (NLP) para a compreensão do contexto nativa e embutida no serviço da Graph API, também usada na plataforma do Messenger.

A Amazon usa também uma solução própria para fazer o reconhecimento de intenções e o contexto, embora não o faça usando apenas algoritmos baseados na nuvem, mas sim com

recurso a *hardware* implantado em produtos lançados como o Echo, que usa a assistente pessoal chamado Alexa.

A Microsoft tem feito o esforço de construir um ecossistema em volta da assistente pessoal Cortana para fornecer um conjunto de dados bem completo com base no *input* fornecido pelos utilizadores do Windows, tornando a compreensão de contexto avançado o suficiente para replicar uma conversa humana e manter a coerência, usando estes dados para dar ênfase a textos falados pela Cortana como resposta aos pedidos dos utilizadores.

2.6.2 Requisitos não funcionais

Neste conjunto de requisitos pode-se avaliar em que medida a facilidade e rapidez das soluções irão afetar a precisão das respostas obtidas no resultado.

2.6.2.1 Facilidade de interação

Todas as soluções propostas são passiveis de um modo de interação com os utilizadores através de agentes externos de conversação, também reconhecidos como canais (*channels*), de comunicação. No entanto, por parte da Microsoft existe um maior conjunto de canais à disposição. Na tabela seguinte tem uma comparação sobre os principais canais, em que soluções podem ser usadas e o seu nível de suporte.

Podemos concluir que existe uma vantagem consideravelmente maior ao escolher o ecossistema de desenvolvimento da Microsoft visto que suporta a grande parte dos canais de comunicação. Para efeitos efetivos na presente solução é a única opção que responde a todas as necessidades.

Tabela 1 – Suporte de canais por solução

Canais de comunicação	Facebook	Amazon	Microsoft
Facebook Messenger	Suporte proprietário	Não é suportado oficialmente	Suporta através de API comum
Facebook WhatsApp	Suporte proprietário	Não é suportado oficialmente	Suporta através de API comum
Skype	Não é suportado	Não é suportado	Suporte proprietário
Echo (dispositivo)	Não é suportado	Suporte proprietário	Não é suportado
Microsoft Teams	Não é suportado	Não é suportado	Suporte proprietário
WeChat	Não é suportado	Não é suportado	Suporta através de API comum
Twilio (SMS)	Não é suportado	Não é suportado	Suporta através de API comum
Cortana	Não é suportado	Não é suportado	Suporte proprietário
Slack	Não é suportado	Não é suportado oficialmente	Suporta através de API comum
Telegram	Não é suportado	Não é suportado	Suporta através de API comum
Via Email	Não é suportado	Não é suportado	Suporta através de API comum
Interface Web	Não é suportado	Não é suportado	Suporta através de API comum

2.6.2.2 Elevado grau de confiança nos resultados

Para uma solução de reconhecimento textual obter sucesso é necessário que tenha um grau de confiança satisfatório nos resultados, baseando em algoritmos eficientes e precisos. Também precisam de conjuntos de dados completos e variados que representem um máximo de combinações úteis para treinar os diferentes aspetos de um modelo analítico de *machine learning*.

O que é capaz de marcar a diferença entre as soluções existentes é essencialmente na quantidade e qualidade dos dados fornecidos.

O Facebook, tendo uma vasta experiência em usar os dados de conversações que provêm do Facebook Messenger, pode contar com milhões de combinações possíveis para melhorar os algoritmos de reconhecimento.

A Amazon, apesar de ter um crescendo volume no mercado, não tem tantos utilizadores com dados relevantes quando comparados com as outras soluções aqui debatidas.

A Microsoft usando o conjunto de telemetria avançada nos seus sistemas operativos, consegue obter um alargado conjunto de dados heterogéneos capazes de proporcionar uma excelente oportunidade nas mais diversas áreas do desenvolvimento de redes neuronais eficientes e eficazes no reconhecimento de padrões.

2.6.3 Considerações

Para a utilização de uma solução que passa pela implementação de um assistente pessoal virtual existem duas categorias criadas por grandes empresas:

1. Ferramentas de desenvolvimento e arquitetura de assistentes pessoais virtuais
    a. Amazon

    Por um lado, a Amazon tem uma forma de recolher informação e interações humanas mais variadas através da plataforma de comércio Amazon, no entanto, as suas ferramentas da *Cloud* não têm tão bons resultados como as da Microsoft e só muito recentemente democratizou as suas ferramentas de desenvolvimento de *bots* pelo que as suas *guidelines* e a sua experiência são ainda limitadas.

b. Microsoft

Relativamente à ferramenta da Microsoft sabe-se que a plataforma de interação utilizada, o *Windows*, é menos variada do que a da Amazon*,* mas tem um número de utilizadores mais elevado, é líder de mercado nas ferramentas da *cloud,* devido à plataforma do Azure, e tem mais anos de investimento e experiência no desenvolvimento de ferramentas para a criação de *bots* e na sua democratização, tendo por isso, *guidelines* mais completas disponíveis para *developers.*

2. Meios ou canais de utilização e consumo de serviços fornecidos por assistentes pessoais virtuais

    a. Messenger do *Facebook*

    Como se pretende que o *bot* seja uma solução integrável em diversos canais de comunicação, esta abordagem não é a mais completa e desejável visto se tratar de apenas um meio de utilização.

    b. Bot Framework da Microsoft

    A solução da Microsoft no que visa aos meios de comunicação são os mais flexíveis visto que é capaz de configurar com vários canais num formato de *plugin*. É compatível com todos os canais principais que os colaboradores estão habituados a interagir.

Para além destas opções existem aplicações e o *site* do CRM que apesar de fazerem o mesmo que os assistentes pessoais virtuais, não o executam de forma tão eficaz, independente e natural e além disso dependem inteiramente da ação humana sendo mais dispendiosas relativamente ao tempo necessário para a sua utilização.

As ferramentas de desenvolvimento escolhidas foram as providenciadas pela Microsoft pois a organização onde este trabalho é desenvolvido tem parceria com a mesma.

3 Design da solução

Pretende-se desenhar uma solução que permita desenvolver um *bot* que comunique com a plataforma do CRM (*Costumer Relationship Manager*) da organização para fazer o registo de *time reports*, compreenda as entidades envolvidas na criação do mesmo e a capacidade de obter a informação necessária do utilizador.

Será usado o *Bot Framework* porque permite uma integração fácil com os meios de comunicação que os colaboradores estão habituados a usar como o Skype e produtos Office 365 com o serviço Bot Connector (Stott, 2016) (Figura 5 – Conjunto de funcionalidades contidas no conector incorporado no SDK).

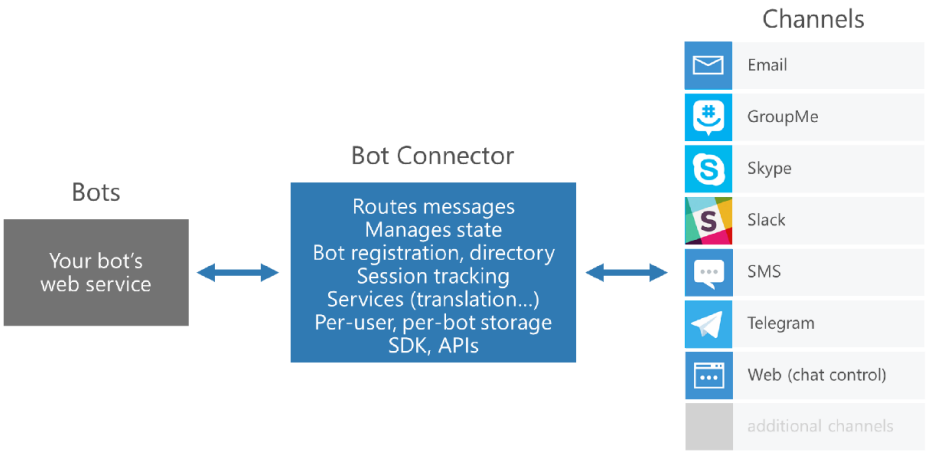

Figura 5 – Conjunto de funcionalidades contidas no conector incorporado no SDK

A forma de desenvolver *bots* inteligentes com esta framework é semelhante ao desenvolvimento de outros produtos desenvolvidos pela organização, pelo se torna mais fácil haver *know how* que possa servir de ajudar no decorrer do desenvolvimento deste trabalho. Por último, a organização está fortemente ligada às tecnologias Microsoft, portanto faz sentido de uma perspetiva estratégica optar por um componente desenvolvido pela mesma.

Para obter uma ideia clara dos cenários que devem ser possíveis nesta solução, foi feita uma análise dos *user stories* aos quais a solução a desenvolver tem de responder.

***User story* 1** – Como colaborador quero poder registar horas

O colaborador dá a intenção que pretende registar horas com input semelhante ao seguinte:

“Estive 4 horas a trabalhar no projeto X”.

O sistema reconhece a intenção e faz as perguntas necessárias para realizar o registo de forma correta e completa, informando o sucesso da operação ao colaborador no final.

***User story* 2** – Como colaborador quero listar os últimos N registos de horas

O colaborador dá a intenção que pretender ver os últimos N registos de horas com um *input* semelhante ao seguinte:

“Quero ver os últimos 5 registos”

O sistema reconhece a intenção e recolhe o parâmetro incluído no *input*. Se não obtiver esta informação é usado como valor por omissão os 5 últimos registos. O sistema realiza a pesquisa pelos registos devolvendo uma listagem dos mesmos ao colaborador.

***User story* 3** – Como assistente pessoal virtual devo alertar o colaborador para o preenchimento do registo de horas.

O assistente pessoal virtual envia uma mensagem ao sistema num momento anterior ao fim da hora de expediente caso tenha detetado que o colaborador tenha falhas no preenchimento das horas nos últimos 7 dias. O sistema comunica com o colaborador.

O diagrama de casos de uso que se segue (Figura 6 – Casos de uso) representa as interações que o colaborador tem com o sistema. É de notar que a capacidade de o assistente pessoal virtual ser proativo em momentos específicos do dia de trabalho faz com que este também seja um ator que interage com o sistema.

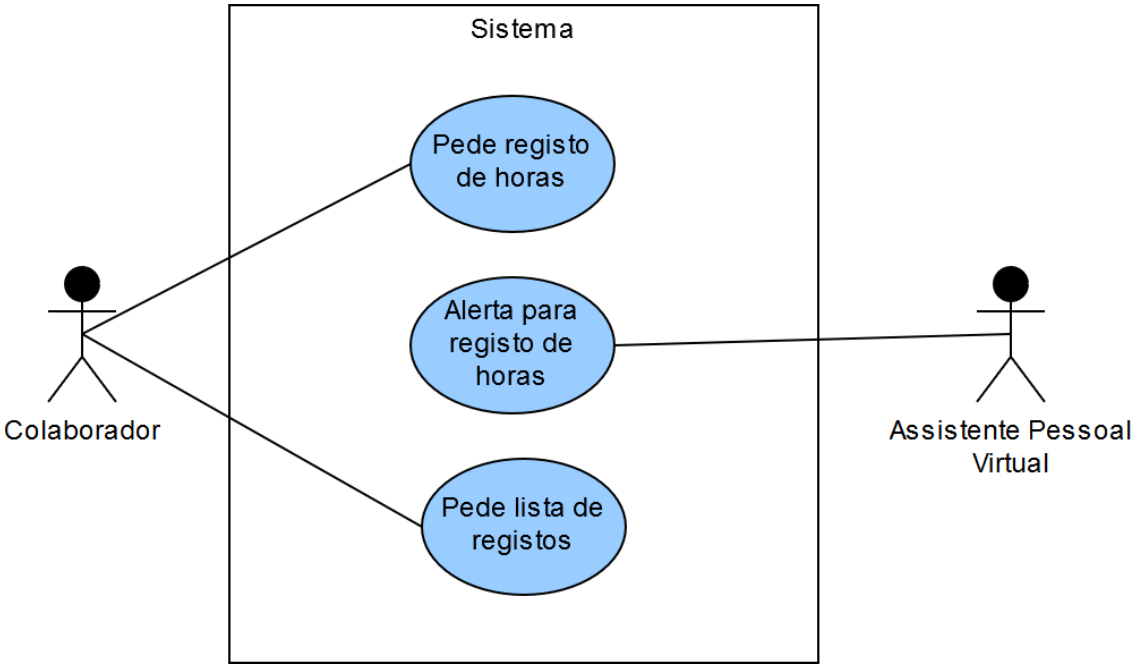

Figura 6 – Casos de uso

Na Figura 7 – Diagrama de sequência da criação de um *time report* usando LUIS, o colaborador usa um canal de comunicação para demonstrar a sua intenção de criar um *time report*. Como exemplo, o canal de comunicação pode o Microsoft Teams e este comunica à Web API o *input* do colaborador. O controlador no *endpoint* da API é primeiramente configurado para usar uma classe responsável pelo diálogo com o serviço LUIS, devolvendo como resultado a intenção que com maior probabilidade de ser a correta. Nesta fase é analisada os parâmetros necessários e, se necessário, o colaborador será questionado sobre todos os parâmetros em falta. Finalizando este passo é, então, possível instanciar um objeto que represente o registo de horas corretamente para posteriormente comunicar a um *data service* que transforma essa informação num modelo compatível com um segundo serviço responsável pela comunicação com o CRM da Microsoft. Este serviço regista o *time report* no CRM e devolve uma resposta de sucesso ao colaborador.

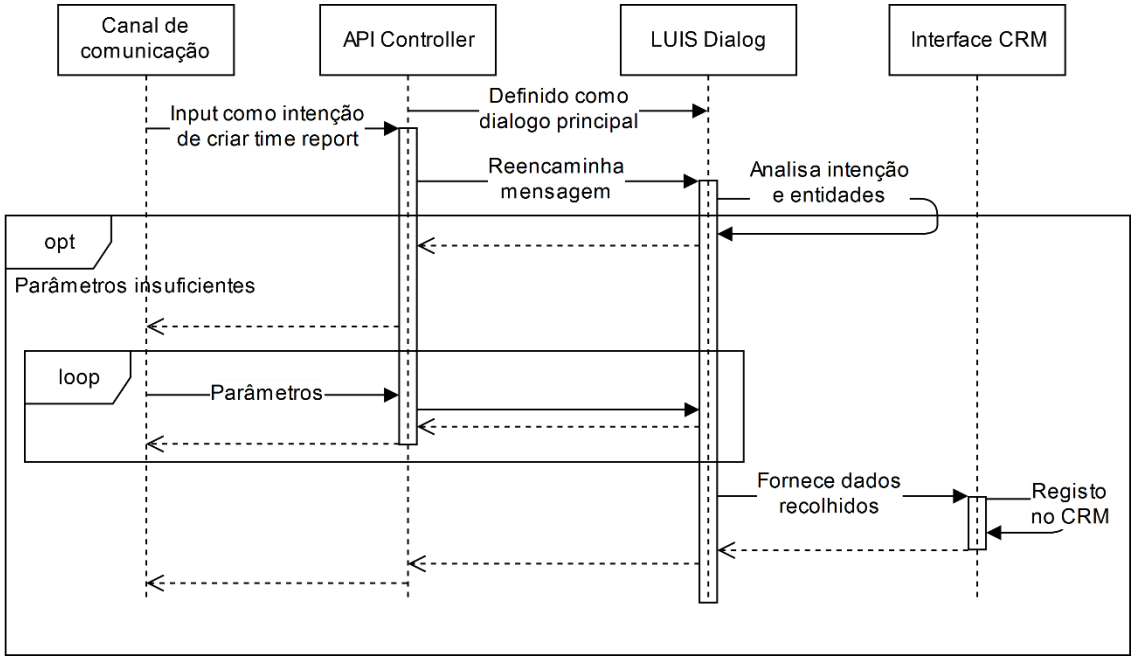

Figura 7 – Diagrama de sequência da criação de um *time report* usando LUIS

A organização de componentes, representada pela Figura 8 – Diagrama de componentes, implica ter como ponto central uma Web API onde estará a implementação do fluxo de conversação do *bot*. Este comunica com o serviço de CRM para controlar as operações CRUD (*Create*, *Read*, *Update* e *Delete*) dos registos de horas. Existe também um componente representativo da base dados onde ficam guardados dados de identificação do colaborador em conjunto com detalhes das interações e registo de atividade do *bot*. O serviço responsável pelo reconhecimento e triagem das intenções e parâmetros de operações do colaborador é identificado na estrutura como o componente LUIS. Por último, o colaborador irá interagir com a solução através de um canal de comunicação que age como uma interface comum para consumir o *bot* exposto pela *Web API*.

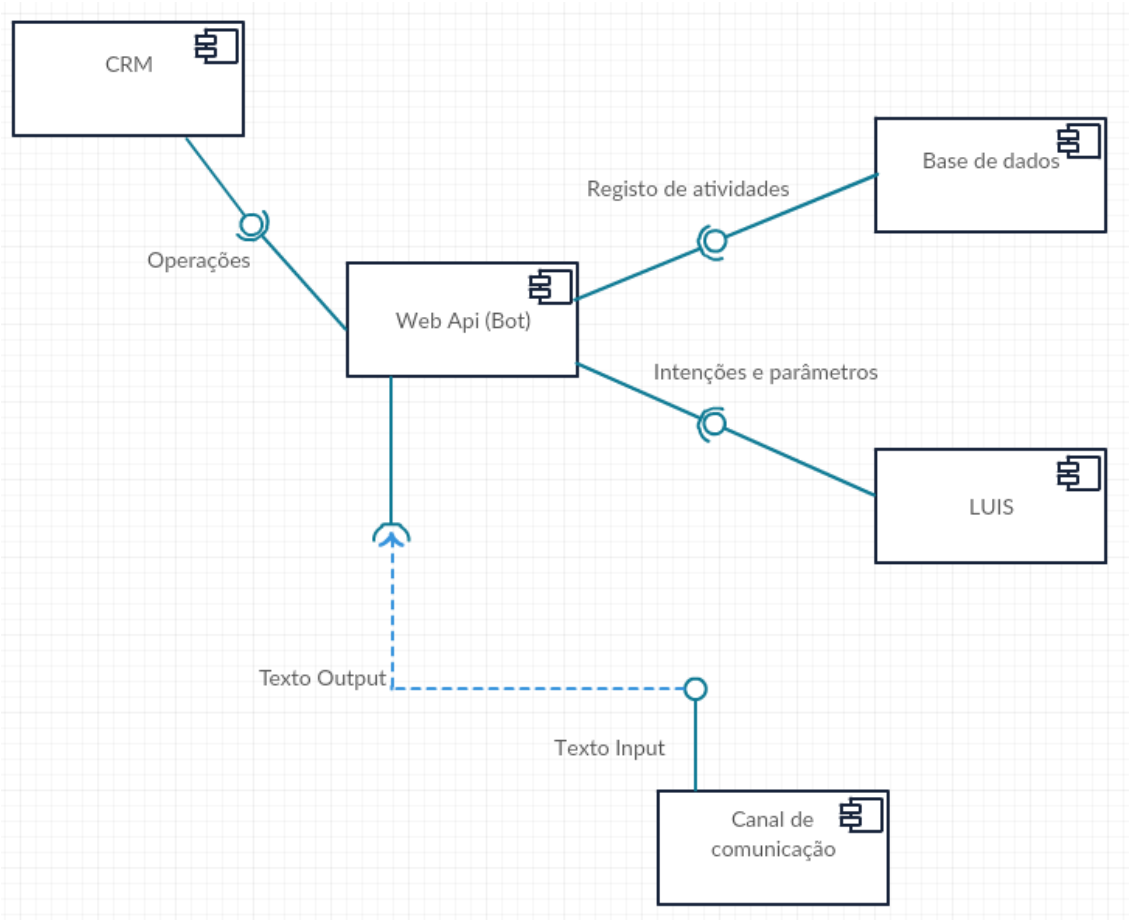

Figura 8 – Diagrama de componentes

Na Figura 9 – Modelo de domínio estão representadas as classes principais e as suas relações.

Um *time report* é composto pela designação do budget e pertence a um utilizador que é o colaborador que dá origem à criação do mesmo. Os budgets são alocados a projetos que por sua vez tem um manager associado.

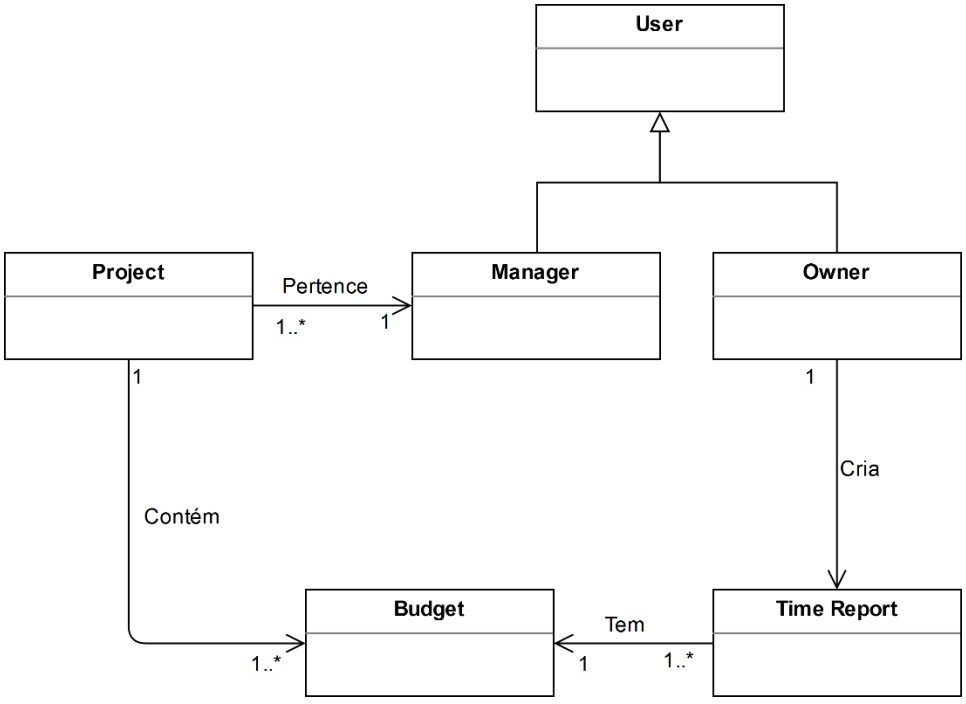

Figura 9 – Modelo de domínio

Esta estrutura é a equivalente à que está a ser utilizada no funcionamento do CRM e, por questões de unificação de conceitos, foi mantida na organização desta solução.

3.1 Ferramentas

O *bot* pode ser desenvolvido em *NodeJS*, usando *JavaScript* como linguagem de programação principal, ou C#. A organização forneceu orientações para que o desenvolvimento da solução fosse feito com C#. A framework .Net é uma das mais usadas nos desenvolvimentos feitos pela organização e permite um ambiente de compilação e *runtime* do código C# (Petrusha, 2017). Adicionalmente, o C# é uma linguagem fortemente tipada (Lippert, 2012) fazendo com que erros de sintaxe na programação sejam praticamente inexistentes. A utilização do *Visual Studio* permite aceder à ferramenta *IntelliSense* que se baseia nesta característica da linguagem e também serve para facilitar e acelerar a escrita de código correto. Com recurso a um *template* para o *Visual Studio*, permite ajudar a começar com uma estrutura que representa as boas práticas na organização deste tipo de projeto.

Como ferramenta para analisar o texto de *input* do utilizador e transformar o mesmo em intenções passiveis de analise pela solução apresentada, irá ser utilizado o LUIS. Esta é a ferramenta principal da *Microsoft* capaz de traduzir linguagem natural em *intents* (intenções) e *entities* (entidades) parametrizáveis de forma a canalizar o que o *bot* pode dar como resposta.

Intents são intenções ou ações desejadas transmitidas através das expressões. Elas compõem a estrutura principal da aplicação e são a primeira linha de triagem do tipo de resposta.

Entities descrevem informações relevantes para a intenção e, podem ser essenciais para ajudar a aplicação a executar a tarefa, por exemplo, o reconhecimento de um contacto de email para agendar uma reunião.

A forma de interagir com o *bot* é feita com recurso a um canal de comunicação. Para a solução proposta irá ser usado o *Microsoft Teams*. Este consiste numa plataforma integrada no Office 365 com o objetivo de interligar colaboradores via conversação que já utiliza o *Azure Active Directory* para os utilizadores de uma organização fazerem *login* e utilizarem o serviço.

De forma a introduzir a capacidade do *bot* ser proativo e avisar o colaborador que tem de realizar time reports ao final do dia - quando se aplica – foi usado o Microsoft Flow.

O Microsoft Flow é uma plataforma que integrar tarefas repetitivas em fluxos de trabalho automatizados entre aplicações e serviços favoritos para obter notificações, sincronizar ficheiros e recolher dados.

No contexto desta solução, o Microsoft Flow vai ser usado para efetuar uma chamada à API do *bot* que é responsável por despoletar a analise de time reports dos colaboradores todos os dias ao final do dia.

3.2 Autenticação

Numa fase inicial, o assistente pessoal não vai saber quem são os colaboradores.

Estes têm de iniciar uma conversa para despoletar o processo de autenticação e consequente identificação.

Após este início, a lógica do *bot* irá dar início ao processo de autenticação (Figura 10 – Processo de SSO (*Single Sign On*)) ao usar uma configuração preparada no Azure AD de forma a fornecer um *token* válido e o qual seja capaz de ser usado para interrogar o Microsoft Graph sobre informação associada ao colaborador. A informação mais relevante que irá ser utilizada é o email. Através do email obtido da autenticação será possível questionar sobre a identificação interna do utilizador no CRM.

O Microsoft Graph é uma API que permite obter informações sobre tudo o que está interligado no Office 365. Com esta API é possível obter informação sobre a pessoal que está autenticada

e o contexto onde se insere. Essas informações incluem o *manager* que poderá ser útil no decorrer do desenvolvimento do assistente pessoal.

A comunicação com o CRM será abstraída do contexto da solução pois será delegada essa responsabilidade à API OData que irá ser consumida. Essa API de serviços do CRM está encapsulada e gerida pelo Azure API Management.

O Azure API Management é um serviço de gestão de APIs capaz de publicar *gateways* de API consistentes e modernos para serviços de back-end existentes alojados em qualquer local, protegendo-os de abusos e sobreutilização. Também é possível obter informação de utilização, estado do funcionamento e adicionar automatismos. Este serviço recorre a um sistema de registo onde os *developers* se podem subscrever e obter chaves de utilização. (Gestão de API, 2018)

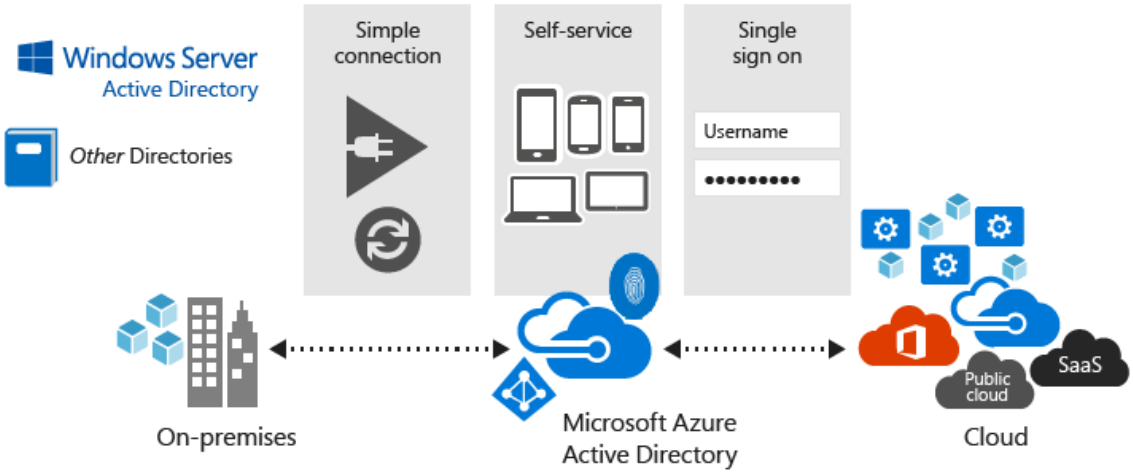

Figura 10 – Processo de SSO (*Single Sign On*)

3.3 CRM

Para haver o registo dos *time reports* é preciso criar um conector com o CRM. Já foi realizada uma API OData que irá servir de ponte com a solução presente neste documento e tem a responsabilidade de abstrair todas as operações que irão ser realizadas diretamente no CRM.

3.4 Persistência

O Bot Builder Framework permite que o *bot* armazene e recupere dados de estado que estão associados a um colaborador, uma conversa ou um colaborador específico dentro do contexto

de uma conversa específica. Os dados de estado podem ser usados para muitos propósitos, como determinar onde a conversa anterior foi interrompida ou simplesmente saudar um usuário que retorna pelo nome. Se armazenar as preferências de um usuário, pode-se usar essas informações para personalizar a conversa na próxima vez que conversar. Por exemplo, o *bot* pode alertar o colaborador para um *time report* que possa querer repetir no dia seguinte.

Para fins de teste e prototipagem, irá ser usado o armazenamento de dados na memória do Bot Builder Framework. Para *bots* de produção, deve-se implementar um adaptador próprio de armazenamento ou usar uma das Azure Extensions. As Azure Extensions permitem que armazenar os dados de estado do *bot* em Table Storage, CosmosDB ou SQL. (Standefer, 2018)

3.5 Resiliência

Todas as operações e transações de informações sensíveis e permanentes devem ser feitas de forma atómica na comunicação com a API OData intermédia do CRM. No caso de eventual falha nesta comunicação a operação é anulada e dada a indicação ao colaborador sobre a situação.

3.6 Ambiente de desenvolvimento

Como ferramenta principal de desenvolvimento é escolhido o *Visual Studio* 2017 por ser uma solução integrada para publicação no *Azure* criando uma aplicação *web*. É também usado um *template* que dá uso ao *Bot Builder* SDK disponibilizado oficialmente pela *Microsoft*.

Para reconhecimento de linguagem natural é criado um projeto LUIS onde é construído o modelo que identifica as entidades necessárias às intenções que se pretende que o *bot* reconheça.

Por último é usado um registo de canais de comunicação no portal do Azure onde será possível gerir a configuração com os mesmo. A configuração da autenticação também tem lugar neste registo.

4 Desenvolvimento da solução

Para construir a solução é preciso preparar as diferentes plataformas e ferramentas que são necessárias.

As diferentes partes podem ser segmentadas pelos seguintes componentes:

- Assistente pessoal virtual
    - LUIS
    - Canais de comunicação
    - *Bot*
        - Diálogos
- Interface CRM (*Costumer Relationship Manager*)
    - Consumo da API OData CRM

Como é possível concluir dos componentes o assistente pessoal virtual é na verdade a combinação do *bot* – que contém a lógica necessária para as diferentes operações –, a ligação com o LUIS – responsável pela categorização dos *inputs* do colaborador em diferentes categorias/intenções – e os canais de comunicação – meios de comunicação com o utilizador.

Para esse conjunto estar completo é preciso registar o *bot* no portal do Azure de forma a obter credenciais de acesso e identificação necessárias para a publicação nos diferentes canais, assim como uma ligação autenticada ao LUIS.

Neste mesmo portal são feitas as configurações para cada canal de comunicação, que tem de ser configurado consoante um conjunto de características próprias a cada um. Para a presente solução, o canal escolhido é o *Microsoft Teams*. Este canal tem uma interligação direta com as tecnologias Microsoft e com a conta usada para o registo do *bot* na plataforma, pelo que não é necessário configurar nada em específico, estando pronto a usar. À exceção dos canais direcionados para integração direta em *web sites*, qualquer outro canal, desde que previamente configurado, tem um endereço que pode ser partilhado com os colaboradores da organização para que o *bot* seja adicionado à lista de contactos.

Um colaborador, ao interagir com o contacto adicionado desta forma está também a consumir a solução pelo intermediário do canal, pois este encaminha a informação da conversa presente com o serviço do *bot*.

Este serviço fica responsável por gerir o fluxo de *input* e *output* da conversa e encaminhar a informação telemétrica para o endereço de *Web Api* exposto pelo *bot*. A partir deste ponto, através do código implementado no *bot*, é chamado o serviço de processamento de linguagem natural (LUIS) para interpretar o *input* recebido. Consoante o resultado desse processamento, é feita uma triagem sobre o rumo da conversa que o *bot* vai executar.

Por último, o *bot* terá de ter a capacidade de ser proativo e efetuar uma verificação que encontre lacunas nos *time reports* e, caso encontre, seja capaz de compilar a informação que deve mostrar. Se o colaborador estiver a interagir com o *bot* neste momento, a conversa terá de ser interrompida temporariamente para alertar sobre as falhas encontradas. A partir desse ponto o utilizador pode escolher tomar uma ação em prol da resolução das falhas ou ignorar e retomar à conversa anterior à interrupção. Este comportamento é possível se houver o conceito de diálogos separados para cada fluxo e uma memória temporária do contexto, separada para diálogo.

4.1 LUIS

Este serviço, desenvolvido sobre o conceito de SaaS, ou *System-as-a-Service*, faz parte de um conjunto de serviços chamado *Cognitive Services* no Azure. Esta denominação deve-se ao conjunto de serviços que a Microsoft fornece aos *developers* para dotarem os seus programas de software com algoritmos avançados de inteligência artificial capazes de ver, ouvir, falar, entender e interpretar os seus utilizadores através de métodos naturais de comunicação. (Cognitive Services, 2017)

Para dar início ao desenvolvimento da app no serviço LUIS, criou-se um projeto no portal do *LUIS* com o nome "*Time Report*". Para isso foi usada uma conta *Microsoft* profissional.

Deve-se usar o portal europeu do LUIS pois é compatível com as regras de proteção de dados do utilizador.

Depois usa-se a entrada do menu "*My apps*" para aceder à listagem de aplicações usadas com a plataforma. Neste ecrã é possível criar uma *app* usando o botão "*New App*". Depois de introduzir as informações do nome da aplicação é possível escolher a linguagem em que o

modelo vai ser adaptado e treinado. Até à altura é possível escolher entre inglês, chinês, francês, espanhol, italiano, alemão, japonês, português do Brasil e coreano. Para o presente trabalho é escolhida a língua portuguesa, pois facilita a interação com o assistente pessoal por colaboradores menos experientes em outras línguas.

De seguida tem de se definir intenções que o modelo analítico deve reconhecer. Para o efeito de operações com *time reports*, existem pelo menos duas intenções a introduzir:

- *CreateTimeReport* – Criar *time report;*
- *ListTimeReport* – Listar *time reports.*

É boa prática dotar o modelo de reconhecer e categorizar à parte todas as outras formas de introdução que não se insiram nos padrões das intenções acima indicadas. Para esse efeito adiciona-se mais uma intenção de *fallback*:

- *None* – nenhuma intenção relevante.

Depois de adicionadas as intenções principal, é aconselhável introduzir dados para treinar o modelo de *machine learning* dando exemplos de frases e expressões que seriam usadas para despoletar cada uma das intenções (Tabela 2 – Treino de intenções no LUIS).

CreateTimeReport	Estive 4 horas no projeto X.
	8 horas projeto Y a fazer resolução de problemas.
ListTimeReport	Qual são os últimos 5 registos?
	O que fiz esta semana?
None	Consegui terminar as tarefas todas.
	O que vou fazer amanhã?

Tabela 2 – Treino de intenções no LUIS

Após este passo, deve-se adicionar entidades através do menu lateral "*Entities*". Para o contexto da criação de *time reports* são criadas as seguintes entidades:

- *ProjectName* – Nome do projeto
- *Context* – Contexto/detalhes do *time report*
- *Date* – Data correspondente
- *Time* – tempo/duração correspondente
- *DateTime* (*Date* + *Time*) – entidade composta pelas entidades simples *Date* e *Time*

Depois de introduzir as entidades, é preciso voltar a ensinar o modelo com novos exemplos de frases e expressões em que é possível reconhecer o uso dessas entidades (Tabela 3 – Treino de reconhecimento de entidades no LUIS).

CreateTimeReport	Estive [4 horas](*Time*) no projeto [X](*ProjectName*).
	[8 horas](*Time*) projeto [Y](*ProjectName*) a fazer [resolução de problemas] (*Context*).
ListTimeReport	Quais são os últimos registos?
	O que fiz [esta semana](*Time*)?
None	Consegui terminar as tarefas todas.
	O que vou fazer [amanhã](*Date*)?

Tabela 3 – Treino de reconhecimento de entidades no LUIS

Quando se despoleta o treino, o serviço reconstrói o modelo de forma mais otimizada, consoante a qualidade e quantidade do conjunto de dados fornecido de forma a construir uma árvore de decisões com pesos atualizados para que processe um futuro texto de input do utilizador e aumente a percentagem de confiança nos resultados obtidos.

Com base nas palavras usadas nas frases incluídas nos treinos, estas são então registadas pela ordem de associação à intenção especificada quando foi classificada manualmente a partir do portal.

No entanto existem dois cuidados a ter quando se procede com o treino do modelo:

- Não criar demasiadas intenções – Quanto mais intenções se adicionar, mais ambíguo poderá ser o resultado para um dado input, obrigando a detalhar melhor os exemplos para cada intenção;
- Fornecer um conjunto equilibrado de exemplos para cada intenção – Se houver um grande desfasamento na quantidade de exemplos entre duas intenções vai causar o modelo ter uma tendência (*bias*) a favor do que tiver mais exemplos. De forma a evitar esse comportamento deve-se dar um número semelhante de exemplos para cada intenção.

Feitos os treinos e testes iniciais ao modelo, deve-se então obter um *endpoint* por onde o serviço vai ser consumido.

Para esse efeito, é preciso gerar uma chave de assinatura e escolher que tipo de ambiente se pretende publicar o *endpoint*. A chave é usada para, a qualquer momento, no código do *bot* poder validar que está a ser usado o modelo correto. O tipo de ambiente para a publicação

pode ser *Staging* ou *Prodution*. Durante todo o desenvolvimento do *bot* foi usado apenas o ambiente de *Staging*.

O *endpoint* final gerado tem a informação relativa à identificação da subscrição do Azure do *developer* e à identificação do modelo. Estas chaves irão ser precisas mais tarde para configurar o código do *bot* de forma a este conseguir comunicar de forma correta com o modelo criado no LUIS e poder consumir o serviço.

Existem outros meios de introduzir conjuntos de dados: o carregamento de ficheiros de texto no formato *json* e pelo próprio consumo do serviço. Em ambos os casos, estes dados são usados para treinar iterativamente o modelo.

O serviço expõe no *endpoint* uma API REST que aceita predominantemente o método GET com o texto de *input* incluído na *query string* sob o parâmetro *q*. O resultado segue uma estrutura *json* que pode ser usada pelo *bot* para executar código dependendo da sua interpretação.

Por exemplo, ao submeter um pedido GET ao *endpoint* com a frase “Estive a trabalhar 4 horas no projeto SmartDocumentor” como input, conseguimos obter resultado como o apresentado pela seguinte estrutura *json*:


```
{
    "query": "estive a trabalhar 4 horas no projeto SmartDocumentor",
    "topScoringIntent": {
        "intent": "CreateTimeReport",
        "score": 0.9957393
    },
  "intents": [
    {
        "intent": "CreateTimeReport",
        "score": 0.9957393
    },
    {
        "intent": "ListTimeReport",
        "score": 0.008134448
    },
    {
        "intent": "None",
        "score": 0.00102361664
    }
  ],
  "entities": [
    {
        "entity": "4 horas",
        "type": "Time",
        "startIndex": 19,
        "endIndex": 25,
        "score": 0.868763
    },
    {
        "entity": "smartdocumentor",
```

```
        "type": "ProjectName",
        "startIndex": 39,
        "endIndex": 53,
        "score": 0.8392739
    }
  ]
}
```

É possível extrair a seguinte informação:

- A intenção que tem maior probabilidade de estar correta é `CreateTimeReport`, traduzindo-se na intenção de criar um registo de horas;
- Existe uma percentagem, embora irrisória, da intenção do utilizador ter sido `ListTimeReport` e `None`;
- Foram reconhecidas duas entidades: `Time` devido ao subtexto "`4 horas`" e `ProjectName` devido ao subtexto "`smartdocumentor`".

4.2 Interface CRM

Para efetuar as ligações necessárias ao CRM da organização é considerado o desenvolvimento de uma WebAPI OData de forma a manter o código coeso e fácil de manter. São usados *endpoints* para expor os métodos disponíveis às operações possíveis de realizar. Esse desenvolvimento já foi realizado no passado e será usado nesta solução.

A WebAPI OData está encapsulada pelo uso do serviço de API Management da Microsoft, que permite ajudar as organizações a publicar APIs para *developers* internos, externos e parceiros, a fim de revelar o potencial do dados e serviços da organização. O API Management fornece as principais competências para garantir um envolvimento do *developer*, telemetria do estado dos negócios, análises, segurança e proteção. (Vlad Vinogradsky, 2018)

Em conjunto com a análise da implementação da API e forma de utilização é criado uma coleção de pedidos com recurso ao Postman. Esta coleção foi desenvolvida em conjunto com a equipa original pela criação do mesmo e a mesma é composta pelo seguinte conjunto de pedidos:

- Obter time reports, budgets, projetos e utilizadores;
- Registar time reports.

Todos os métodos são chamados através de pedidos POST com o *header* "Ocp-Apim-Subscription-Key" onde é indicada a chave de subscrição fornecida pelo portal de registo, necessária para consumir os serviços prestados.

É também obrigatório incluir no corpo da mensagem a informação em formato json com o token que é definido pela API e é usado como medida de segurança de forma a evitar que a API aceite pedidos de fontes desconhecidas e inseguras.

Para o caso dos pedidos de obtenção de informações, é necessário incluir um parâmetro no corpo da mensagem relativo ao *endpoint* da API do CRM que se dá pelo nome *entity* e serve para indicar que parte do CRM está a questionar a informação. Tem também suporte para filtros OData que podem ser indicados nesta propriedade.

Um exemplo de pedido de entidade que procure retornar os *time reports* para um dado utilizador para um dado projeto segue a seguinte lógica e sintaxe:

"/dev_timesheets?**$filter=_dev_projectid_value eq** <GUID projeto> and **_ownerid_value eq** <GUID utilizador> &**$top**=10&**$orderby=createdon desc**"

Após tomar conhecimento do funcionamento integral da API, dá-se lugar ao desenvolvimento da componente que irá servir de interface para os pedidos a serem feitos.

Para garantir que são desenvolvidos os métodos certos para todos os cenários envolvidos nesta solução, é adotada a técnica de TDD (*Test Driven Development*) para o desenvolvimento da interface comum com o CRM e como tal são definidos os seguintes métodos presentes na Figura 11 fazendo correspondência com as funcionalidades que a interface terá de expor para posteriormente ser usada pelo *bot* propriamente dito.

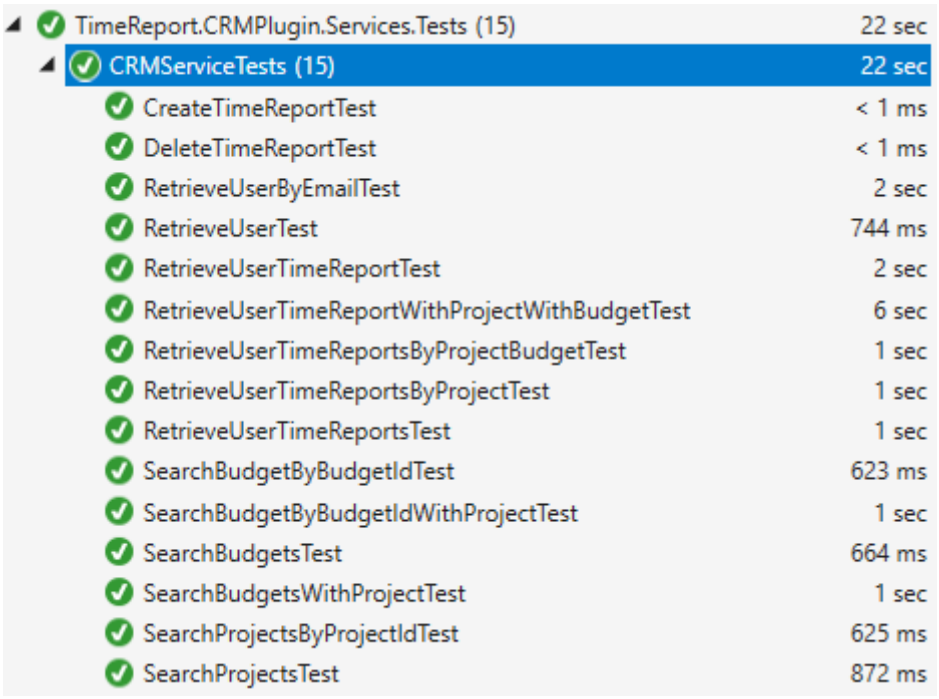

Figura 11 – Testes a cada funcionalidade exposta pela interface CRM

Para dar lugar ao desenvolvimento da interface que dará resposta ao que está definido pelos testes, é então criado um projeto de Visual Studio do tipo *Class Library* (Figura 12 – Estrutura projeto que contém o plugin para CRM) onde é determinada a estrutura dos modelos a serem usados para compreender a resposta dos pedidos de obtenção de informação e também dos modelos que sigam a estrutura do que é pedido pela API no momento de se criar um *time report*.

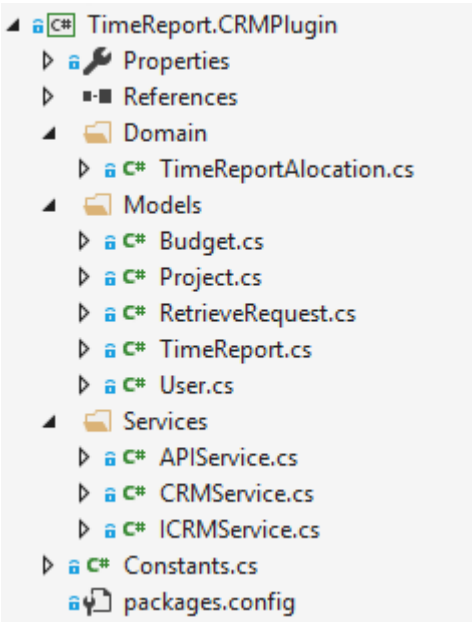

Figura 12 – Estrutura projeto que contém o plugin para CRM

A parte principal deste projeto é a interface que expõe e a declaração dos métodos (Figura 13 – Declaração da interface dos serviços de comunicação ao CRM).

Os métodos implementados compreendem a obtenção de utilizadores, projetos, *budgets*, *time reports* e as operações de escrita que envolvem criar e apagar time reports.

```
public interface ICRMService
{
    /// <summary> Creates a time report entity on CRM and returns the id associated  ...
    Task<string> CreateTimeSheet(TimeSheet timeSheet);

    /// <summary> Deletes a time report with the given the id
    Task<bool> DeleteTimeSheet(string timeSheetId);

    /// <summary> Retrieves the projects based on search or a project based on his i ...
    Task<IEnumerable<Project>> RetrieveProjects(string project = null, bool includeOwner = false, int nTop
      = 10);

    /// <summary> Retrieves the budget based on project search and search term for b ...
    Task<IEnumerable<Budget>> RetrieveBudgets(string project, string searchText = null, bool includeProject
       = false);

    /// <summary> Retrieves the budget based on his id
    Task<Budget> RetrieveBudget(string budget, bool includeProject = false);

    /// <summary> Retrieves a enumerable of time reports that satisfy the search par ...
    Task<IEnumerable<TimeSheetRetrieve>> RetrieveTimeSheets(string user, string project = null, string
      budget = null, bool includeProject = false, bool includeBudget = false, int nTop = 10, DateTime?
      startDate = null, DateTime? endDate = null, string sort = null);

    /// <summary> Retrieves a time report based on his id
    Task<TimeSheetRetrieve> RetrieveTimeSheet(string timeSheetId, bool includeProject = false, bool
      includeBudget = false);

    /// <summary> Retrieves the user based on his id
    Task<User> RetrieveUser(string userId);

    /// <summary> Retrieves the user based on his email
    Task<User> RetrieveUserByEmail(string email);
}
```

Figura 13 – Declaração da interface dos serviços de comunicação ao CRM

4.3 *Bot*

A Microsoft tem um sistema de repositório e distribuição de pacotes de código e bibliotecas .Net (Figura 14 – Esquema de distribuição de NuGet Packages), também conhecidos como Nuget Packages, para ajudar e acelerar o desenvolvimento de aplicações e serviços usando o IDE Visual Studio (Nandwani, 2018).

Qualquer *developer* e organização pode começar por criar um projeto com o *source code* para posteriormente gerar um package que pode então ser publicado para o repositório central *online*. A partir desse ponto, a comunidade de *developers* pode instalar esse package nos seus projetos .Net e usufruir das funcionalidades implementadas, podendo interagir através de interfaces expostas.

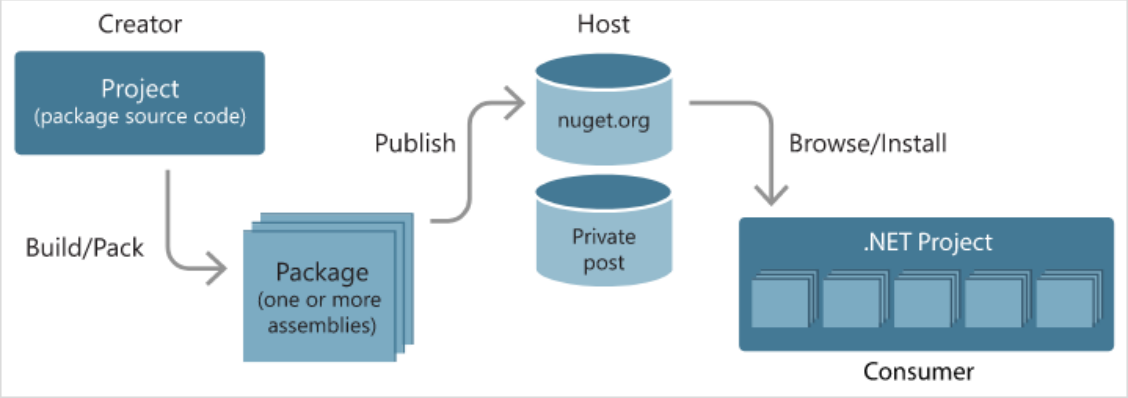

Figura 14 – Esquema de distribuição de NuGet Packages

Para o desenvolvimento do *bot* é preciso especialmente do NuGet *package* Microsoft.Bot.Builder. Este package contém o SDK e o conector para utilizar um conjunto de funcionalidades de controlo do fluxo de conversação característicos.

Para acelerar o processo de criação de projeto com os *packages*, é fornecido pela Microsoft um *template* de criação de projeto (Figura 15 – Estrutura da solução recomendada pelo *template*) que contém os pacotes Nuget necessários. Adicionalmente, também é demonstrada uma estrutura que segue boas práticas de código.

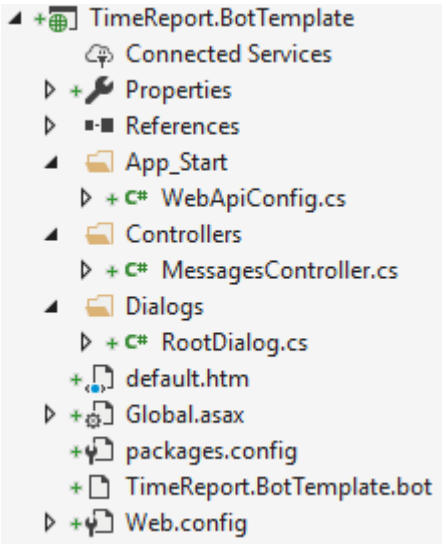

Figura 15 – Estrutura da solução recomendada pelo *template*

Apesar do *template* adiantar uma grande parte do trabalho inicial, é necessário atualizar os pacotes *NuGet* para a sua versão mais recente (Figura 16 – Atualização dos packages *NuGet* do projeto) de forma não perder as correções e melhorias que são constantemente lançadas.

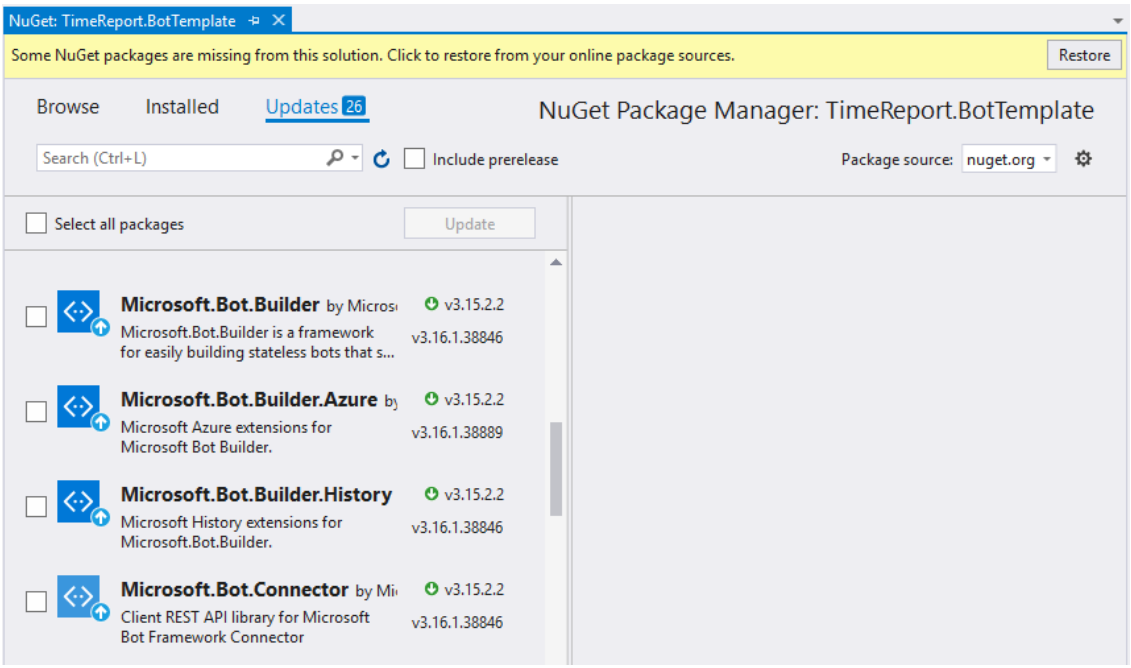

Figura 16 – Atualização dos packages *NuGet* do projeto

A organização do *template* é muito semelhante à de um projeto de Web API desenvolvida em C#. Existe o conceito de rotas compostas por controladores que ficam responsáveis pelos métodos a serem expostos pela API.

Neste caso existe o controlador chamado *MessagesController* (Figura 17 – Implementação do controlador principal de invocação do *bot*) que expõe um *endpoint* POST. Este é o ponto de entrada comum para interagir com o *bot* e a comunicação é feita através do conetor do Bot Framework que recebe um objeto do tipo *Activity*. Esta classe advém do *namespace Microsoft.Bot.Connector* que está incluído no pacote *NuGet Microsoft.Bot.Builder*.

```
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Connector;

namespace TimeReport.BotTemplate
{
    [BotAuthentication]
    0 references | 0 changes | 0 authors, 0 changes
    public class MessagesController : ApiController
    {
        /// <summary>
        /// POST: api/Messages
        /// Receive a message from a user and reply to it
        /// </summary>
        0 references | 0 changes | 0 authors, 0 changes
        public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
        {
            if (activity.GetActivityType() == ActivityTypes.Message)
            {
                await Conversation.SendAsync(activity, () => new Dialogs.RootDialog());
            }
            else
            {
                HandleSystemMessage(activity);
            }
            var response = Request.CreateResponse(HttpStatusCode.OK);
            return response;
        }
```

Figura 17 – Implementação do controlador principal de invocação do *bot*

Quando o utilizador interage com um canal de comunicação, escrevendo um texto ou interagindo sob outro tipo de ação para ser analisado pelo sistema, é gerado, pelo conetor que fornece a ligação entre canal e Web Api disponibilizada pelo *bot*, uma atividade que representa o estado da conversa incluindo o input dado pelo colaborador através do canal.

Uma atividade é um conceito que representa a informação que flui do utilizador para o *bot* e vice-versa, contendo o conteúdo das mensagens adicionando todo um conjunto de informação que pode ser útil para enriquecer a experiência que o *bot* pode fornecer.

O *Bot Framework* usa este objeto para comunicar mudanças na conversa ao *bot*, agindo assim como *event handler*. Os tipos de atividades compreendem a chegada de uma mensagem, mudança na relação com o contacto, atualização da conversa, indicação que o utilizador está a escrever, aviso de chegada de comunicação (*ping*), fim de conversa, desencadeamento de atividade, evento, invocação, informação de remoção dos dados do utilizador, atualização na instalação e reação a alguma mensagem.

Neste objeto é verificado se o tipo de atividade é uma mensagem, o que indica que é proveniente do utilizador. Neste caso é preciso processar a mesma. A estrutura de um projeto de *bot* compreende a utilização de diálogos, uma forma potente e flexível de lidar conversas

guiadas onde facilita a obtenção de indicações para tratar um assunto específico. (Brandl, Standefer, Vo, Dempsey, & Delimarsky, 2017) Portanto, se a atividade compreender uma mensagem do utilizador para ser analisada, irá então ser usado um diálogo para dar início à conversa. Este diálogo é terá como nome nesta solução como RootDialog.

Criou-se vários diálogos para representar uma serie de acontecimentos que têm navigabilidade implícita, tornando a definição do contexto mais clara para o utilizador que está a interagir com o *bot*.

A forma de navegar entre diálogos funciona sob uma *stack*. Quando o utilizador interage primeiramente com o assistente pessoal é inicializado o seguinte fluxo de diálogos:

RootDialog - Sempre que se dá início a uma conversa esta diálogo é o primeiro na *stack* a entrar em ação. Tem a responsabilidade de invocar o diálogo responsável pela autenticação para garantir que o utilizador tem identidade compatível com o serviço que é disponibilizado por este assistente pessoal. Após este passo é reencaminhada a mensagem original para o diálogo capaz de reconhecer a intenção da mensagem original.

AuthDialog - Diálogo responsável pela autenticação. Se o utilizador ainda não estiver autenticado, é espoletado um pedido de autenticação num fornecedor externo ao ambiente da conversa onde é permitido usar o tipo de autenticação que o colaborador tem definida na sua conta. Este processo serve para abstrair fluxos adicionais caso, por exemplo, o colaborador tenha verificação em dois passos. Desta forma, quando o fluxo de autenticação acaba é devolvido o *token* para o *bot* guardar e utilizar essa informação para obter o email interno do colaborador na organização e usar essa informação para questionar a interface de CRM de forma a obter o ID interno do colaborador na plataforma de CRM. O email em conjunto com um *token* fornecido pela equipa que criou a API OData de ligação ao CRM são as informações necessárias para efetuar as operações de registo dos *time reports* em nome do colaborador ao longo da sua utilização do *bot*. Essa informação é guardada na memória de contexto que é mantida pela framework.

ProActiveDialog - Diálogo responsável pela análise de falhas nos registos do colaborador nos últimos dias e, caso se confirme haver falhas, deve haver um alerta da situação. O colaborador também pode tomar ação sobre este alerta no sentido de colmatar as falhas (Figura 18 – Código responsável pela verificação de falhas nos registos do colaborador).

```
int nDays = 7;

ICRMService crmService = new CRMService();

if (context.UserData.ContainsKey("CRMInternalEmailAddress")
    && context.UserData.ContainsKey("CRMInternalUserId"))
{
    var userId = context.UserData.GetValue<string>("CRMInternalUserId");
    var userEmail = context.UserData.GetValue<string>("CRMInternalEmailAddress");

    TimeSheetsFound = (await crmService.RetrieveTimeSheets(userId,
        includeProject: true,
        includeBudget: true,
        startDate: DateTime.UtcNow.Subtract(new TimeSpan(nDays, 0, 0, 0)).Date,
        endDate: DateTime.UtcNow))?
        .ToList();

    var timeSheetsMissing = AnalizeTimeSheets(context, TimeSheetsFound, nDays);

    await context.SayAsync($"Encontrei {timeSheetsMissing.Count} falhas de registos " +
        $"de horas nos ultimos {nDays} dias. Dão um total de " +
        $"{timeSheetsMissing.Sum(t => int.Parse(t.Hours))} horas não reportadas.");

    this.TimeSheetsMissing = timeSheetsMissing;

    PromptDialog.Confirm(
                    context,
                    ResumeShowTimeReport,
                    $"Queres preencher estas falhas?",
                    "Não percebi, tenta outra vez por favor.",
                    options: new string[] { "Sim", "Não" },
                    patterns: new string[][] {
                        new string[] { "sim", "ok", "Sim", "OK", "Ok" },
                        new string[] { "não", "nao", "nah", "Não" } },
                    promptStyle: PromptStyle.Auto);
}
else
    context.Done(true);
```

Figura 18 – Código responsável pela verificação de falhas nos registos do colaborador

É feito um controlador de API dispõe de um *endpoint* adicional (Figura 19 – Implementação do controlador que aciona o comportamento proativo do *bot*) que pode acionar a componente proativa do assistente pessoal, invocando este diálogo correspondente.

```
namespace TimeReport.Bot.Controllers
{
    public class ProActiveController : ApiController
    {
        [HttpGet]
        public async Task<HttpResponseMessage> Activate()
        {
            try
            {
                if (!string.IsNullOrEmpty
                  (ConversationStarter.conversationReference))
                {
                    //We don't need to wait for this, just want to start the
                    interruption here
                    await ConversationStarter.Resume();

                    var resp = new HttpResponseMessage(HttpStatusCode.OK);
                    resp.Content = new StringContent(
                        $"<html><body>Message sent, thanks.</body></html>",
                        System.Text.Encoding.UTF8,
                        @"text/html");
                    return resp;
                }
                else
                {
                    var resp = new HttpResponseMessage(HttpStatusCode.OK);
                    resp.Content = new StringContent(
                        $"<html><body>You need to talk to the bot first so " +
                        $"it can capture your details.</body></html>",
                        System.Text.Encoding.UTF8,
                        @"text/html");
                    return resp;
                }
            }
            catch (Exception ex)
            {
                return Request.CreateErrorResponse(HttpStatusCode.BadRequest,
                  ex);
            }
        }
    }
}
```

Figura 19 – Implementação do controlador que aciona o comportamento proativo do *bot*

Este controlador é invocado por um fluxo criado no Microsoft Flow que por sua vez é configurado para executar uma chamada GET a este *endpoint* (Figura 20 – Configuração do fluxo no Microsoft Flow).

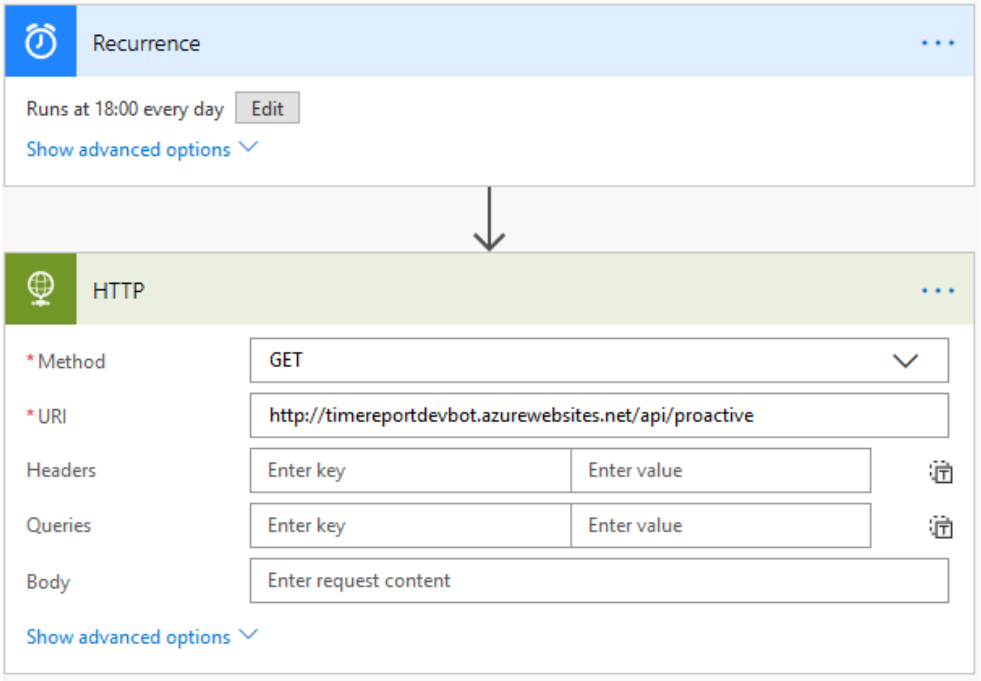

Figura 20 – Configuração do fluxo no Microsoft Flow

No controlador, é invocado um método que provoca uma interrupção na conversa atual, guarda o estado e inicia uma nova *stack* de diálogos com a inicialização do diálogo ProActiveDialog (Figura 21 – Implementação do código que interrompe a conversa atual e dá início a outra).

```
public class ConversationStarter
{
    public static string conversationReference;

    //This will interrupt the conversation and send the user to ProActiveDialog, then wait until
      that's done
    public static async Task Resume()
    {
        var message = JsonConvert.DeserializeObject<ConversationReference>
          (conversationReference).GetPostToBotMessage();
        var client = new ConnectorClient(new Uri(message.ServiceUrl));

        using (var scope = DialogModule.BeginLifetimeScope(Conversation.Container, message))
        {
            var botData = scope.Resolve<IBotData>();
            await botData.LoadAsync(CancellationToken.None);
            var task = scope.Resolve<IDialogTask>();

            //interrupt the stack
            var dialog = new ProActiveDialog();
            task.Call(dialog.Void<object, IMessageActivity>(), null);

            await task.PollAsync(CancellationToken.None);

            //flush dialog stack
            await botData.FlushAsync(CancellationToken.None);
        }
    }
}
```

Figura 21 – Implementação do código que interrompe a conversa atual e dá início a outra

ListTimeReportDialog - Diálogo responsável por obter os últimos registos de *time reports*. É usada a identificação interna do utilizador para invocar o método da interface com o CRM que devolva os 10 registos mais recentes. Este valor é configurável no plugin.

Depois de obter a listagem é feita uma análise de padrões capaz de tornar a informação mais sucinta para ser apresentada ao colaborador. Um exemplo claro de tal analise é capaz de reconhecer que um mesmo projeto e budget foram registados 3 dias com durações iguais de 8 horas no decorrer da listagem recebida pelo CRM e apresentar um cartão de informação a representar:

"Esteve a trabalhar Projeto X | Budget Y nos dias 1,2 e 5 de fevereiro de 2018"

Nos casos em que tal analise não retorne formas mais resumidas de apresentar a informação, é então usada paginação para não sobrecarregar o espaço na conversa.

No final deste diálogo, este é descartado da *stack* retomando ao diálogo de reconhecimentos de intenções de forma a haver nova triagem de ações consoante a interação seguinte.

LUISDialog – Como esta classe será usada para efetuar a comunicação com o LUIS é preciso que se garanta que é passível de ser serializado para que se possa transformar a mesma num correspondente tipo de dados compatível com o que será enviado para o LUIS. Na resposta que advém do serviço irá estar incluída a intenção principal reconhecida e esta terá de ser avaliada por um método a ser desenvolvido nesta classe (Figura 22 – Configuração do LUIS com métodos para cada intenção). Para isso, são indicados métodos para cada intenção usando o atributo *LuisIntent* que recebe como parâmetro uma *string* com descritivo da intenção correspondente ao equivalente criado na aplicação LUIS. Desta forma é garantido que um dado método irá ser executado consoante a intenção que se reconheça como principal ou, em termos de estatísticos, a mais provável e próxima da intenção verdadeira passada pelo colaborador.

Quando cada método atribuído a intenções do LUIS é invocado irá receber como parâmetros o contexto da conversa e o resultado do LUIS.

```
namespace TimeReport.Bot.Dialogs
{
    [Serializable]
    public class LUISDialog : LuisDialog<object>
    {
        public LUISDialog() : base(new LuisService(new LuisModelAttribute(
            ConfigurationManager.AppSettings["LUISApplicationID"],
            ConfigurationManager.AppSettings["LUISAuthoringKey"],
            domain: ConfigurationManager.AppSettings["LuisAPIHostName"])))
        {
        }

        #region None

        [LuisIntent("None")]
        public async Task None(IDialogContext context,
          IAwaitable<IMessageActivity> activity, LuisResult result)...

        #endregion

        #region ListTimeReport

        [LuisIntent("ListTimeReport")]
        public async Task ListTimeReport(IDialogContext context,
          IAwaitable<IMessageActivity> activity, LuisResult result)...

        private Task ResumeAfterListTimeReportDialog(IDialogContext context,
          IAwaitable<object> result)...

        #endregion

        #region CreateTimeReport

        [LuisIntent("CreateTimeReport")]
        public async Task CreateTimeReport(IDialogContext context,
          IAwaitable<IMessageActivity> activity, LuisResult result)...

        private async Task ResumeAfterCreateTimeReportDialog(IDialogContext
          context, IAwaitable<object> result)...

        #endregion
```

Figura 22 – Configuração do LUIS com métodos para cada intenção

Para a presente solução, é incluído um primeiro método que irá executar quando não é reconhecida nenhuma intenção, designando o atributo *LuisIntent* com o parâmetro *None*. Neste método é analisado qual a segunda intenção mais provável e indica ao utilizador sugestões do que este terá tentado indicar.

Num segundo método, é implementado o fluxo de trabalho do *bot* quando o LUIS reconhece que a intenção do utilizador é listar *time reports* usando o atributo *LuisIntent* com o parâmetro *ListTimeReport*. Neste método, o CRM da organização irá servir para pesquisar e obter informação de *time reports* que estejam registados para ser organizados em forma de lista para posterior apresentação ao colaborador. No resultado proveniente do LUIS é possível verificar se o colaborador incluiu alguma indicação temporal pela qual pretende filtrar os resultados da

listagem que pretende. Nesse caso, é incluída na pesquisa os filtros solicitados de forma a que o CRM traga a informação relevante ao que é pedido.

Num terceiro método, irá ser tratado o fluxo de criação de *time reports* usando o atributo *LuisIntent* com o parâmetro *CreateTimeReport*. Neste método é avaliado um conjunto de parâmetros adicionais que o colaborador possa ter incluído no seu pedido original que irá servir para inicializar um objeto referente ao *time report*. Posteriormente a esta análise inicial, a conversa em encaminhada para o diálogo responsável pela criação de *time reports* chamada *CreateTimeReportDialog*.

CreateTimeReportDialog – Diálogo responsável por obter toda a informação necessária para o registo de um *time report*.

Na eventualidade da informação fornecida ser insuficiente, o *bot* irá dar início a um conjunto de questões sobre as propriedades que faltam para que o registo seja realizado com sucesso. A este processo é chamado *FormFlow* (Figura 23 – Invocação do *Form Dialog* para o preenchimento do objeto de *time report*) e que compreende uma classe com as propriedades necessárias para um *time report* e que contêm o atributo *Prompt* onde recebe como parâmetro o texto a ser mostrado ao utilizador aquando do pedido pela referida informação (Figura 24 – Exemplo de propriedades da classe representativa de um *time report* com o atributo *Prompt*).

```
var formDialog = new FormDialog<TimeSheetModel>(_timeSheet,
  _timeSheet.BuildForm);

context.Call(formDialog, this.ShowOptions);
```

Figura 23 – Invocação do *Form Dialog* para o preenchimento do objeto de *time report*

Quando o utilizador introduz a informação relativa a uma dada entidade, esta é validada consoante a serialização da mesma e o tipo de dados que caracteriza a propriedade. Se o tipo de dados da propriedade relativa à designação do número de dias que o colaborador trabalhou num dado projeto for um número inteiro, este não poderá introduzir outro tipo de input senão um número inteiro. Desta forma a gestão de obtenção de informação por parte do *bot* é mais rápida e eficaz, tanto na fase de desenvolvimento como na fase da utilização do mesmo.

```
[Serializable]
public class TimeSheetModel : TimeSheet
{
    public TimeSheetModel() { }

    public TimeSheetModel(TimeSheet timeSheet)...

    [Prompt("Qual é o projeto?")]
    public string ProjectSearch { get; set; }

    [Prompt("Quantas horas estiveste nesta tarefa?")]
    public string Hours { get; set; }

    [Prompt("Quando estiveste nesta tarefa?")]
    public string Date { get; set; }

    [Prompt("Queres acrescentar alguma descrição? (Sim ou não)")]
    public string HasReason { get; set; }

    [Prompt("Qual a descrição?")]
    public string Reason { get; set; }
```

Figura 24 – Exemplo de propriedades da classe representativa de um *time report* com o atributo *Prompt*

No final deste questionário usando o FormFlow, o *bot* apresenta automaticamente uma retrospetiva das respostas dadas com o intuito de validar e confirmar as mesmas. Finalmente, este processo irá retornar um objeto com toda a informação necessária para registar o *time report* com sucesso no CRM de volta ao diálogo CreateTimeReportDialog. Por sua vez, este diálogo deve perguntar se o colaborador pretende dar continuidade à criação do *time report* (Figura 25 – Pergunta para confirmação de criação de time report).

```
private async Task ShowOptions(IDialogContext context,
  IAwaitable<TimeSheetModel> result)
{
    _timeSheet = await result;

    PromptDialog.Confirm(
                context,
                ResumeCreateTimeReport,
                $"Confirmas a criação deste time report?",
                "Não percebi, tenta outra vez por favor.",
                options: new string[] { "Sim", "Começar de
        novo" },
                patterns: new string[][] { new string[]
        { "Continuar", "siga", "continuar", "s", "y", "sim", "ok",
         "Sim", "OK", "Ok" }, new string[] { "Começar de novo",
        "começar", "começar de novo", "comecar", "comecar de
        novo", "novo", "não", "nao", "nah", "Não" } },
                promptStyle: PromptStyle.Auto);
}
```

Figura 25 – Pergunta para confirmação de criação de *time report*

Se a resposta do colaborador for positiva é então validado uma última vez se o objeto está devidamente preenchido e é então entregue aos serviços que irão comunicar o registo ao CRM (Figura 26 – Código responsável pela criação de *time report* consoante resposta do colaborador).

```
private async Task ResumeCreateTimeReport(IDialogContext context,
  IAwaitable<bool> result)
{
    var response = await result;

    if (response && _timeSheet != null)
    {
        if (ValidateTimeSheet(_timeSheet.ToTimeSheet()))
        {
            await Extensions.ConversationHelpers.SendTyping
            (context.Activity as Activity);

            ICRMService crmService = new CRMService();

            string newTimeSheetId = await crmService.CreateTimeSheet
            (_timeSheet.ToTimeSheet());

            if (string.IsNullOrEmpty(newTimeSheetId))
            {
                await context.SayAsync("Houve um erro a tentar criar o
            Time Report. Tenta novamente.");
                context.Done(false);
            }

            await context.SayAsync("Time Report criado com sucesso");

            context.Done(_timeSheet.ToTimeSheet());
        }
        else
        {
            await context.SayAsync("Houve um erro a tentar criar o
            Time Report. Tenta novamente.");

            await StartAsync(context);
        }
    }
    else
    {
        await context.SayAsync("Vamos começar de novo então.");

        var formDialog = new FormDialog<TimeSheetModel>(new
          TimeSheetModel(), _timeSheet.BuildForm);

        context.Call(formDialog, this.ShowOptions);
    }
}
```

Figura 26 – Código responsável pela criação de *time report* consoante resposta do colaborador

No final deste processo, este diálogo é descartado da *stack* e é retomado o diálogo com integração ao LUIS.

Em resumo, a estrutura do projeto que contempla toda a lógica relacionada com o *bot* contém a estrutura de um projeto Web API (Figura 27 – Estrutura do projeto que contém o *bot*) com:

controladores necessários para a comunicação com o Bot Connector e componente de ativação remota; os diálogos para dar resposta a cada funcionalidade a que se compromete responder, incluindo o dialogo que invoca o serviço LUIS para encaminhar a conversa para um subconjunto de diálogos mais apropriados e utilização da interface de comunicação com o CRM.

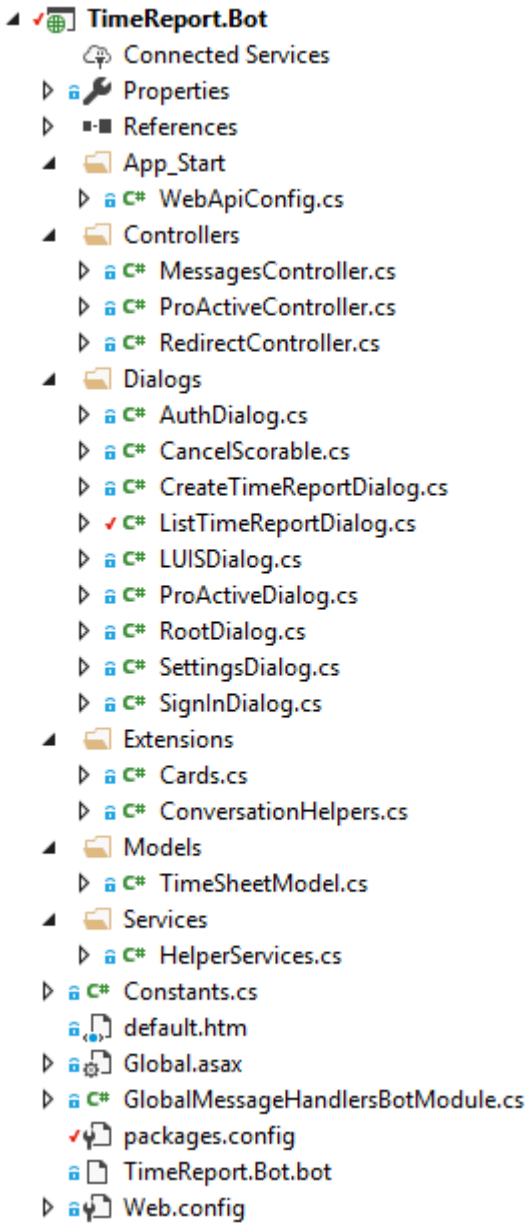

Figura 27 – Estrutura do projeto que contém o *bot*

4.4 Canais de comunicação

Devido à natureza da interação entre os canais de comunicação e o *bot* serem a partir de uma Web API, a solução criada é reconhecida como uma Web App com uma API exposta.

Para efeitos de investigação e testes, a publicação é feita sob a configuração de desenvolvimento (Figura 28 – Publicação em ambiente de Dev).

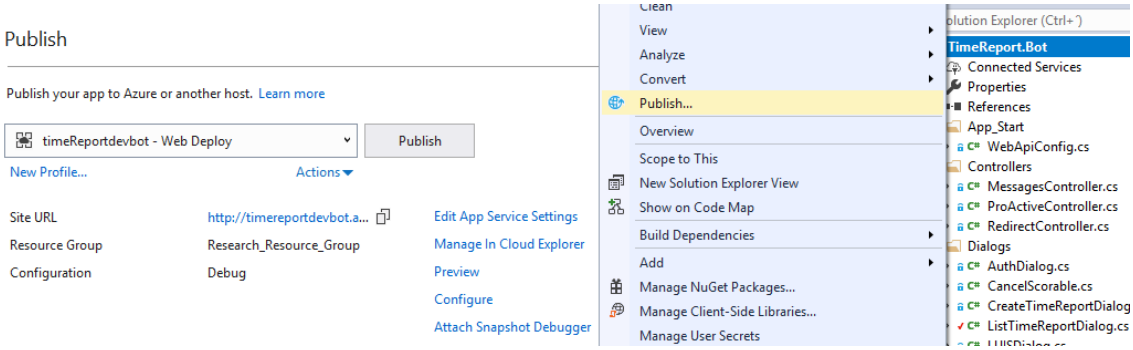

Figura 28 – Publicação em ambiente de Dev

No Azure, a publicação de uma Web App implica usar um plano de App Service tipicamente usado para publicação de *Web Apps* (Figura 29 – *Wizard* para publicação).

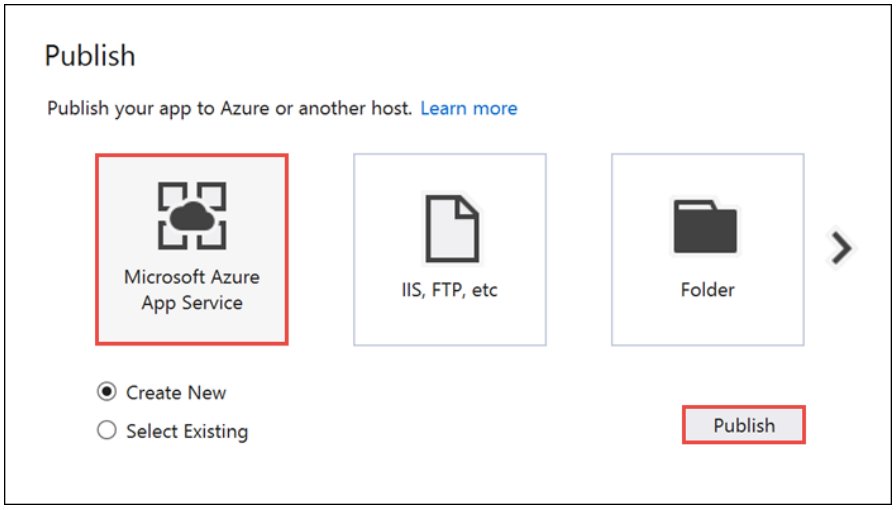

Figura 29 – *Wizard* para publicação

Apesar do processo de criação e publicação ser simples, o que está realmente a acontecer é a compilação do código do(s) projeto(s) envolvidos da solução e todo o processo de aprovisionamento desde máquinas virtuais geridas e hospedadas no *Azure*, instalação dos componentes necessários como o IIS (*Internet Information Server*), publicação do conteúdo compilado, neste caso do *bot*, e ativação do endereço de acesso público para o mesmo.

O IIS é um servidor web flexível e seguro para hospedar qualquer tipo de conteúdo na *Web*. Da transmissão de média até aplicações *web*, o IIS é escalável e a sua arquitetura está pronta para lidar com tarefas exigentes. (A flexible & easy-to-manage web server, 2018)

Após o processo de publicação estar concluído, é apresentada a página inicial da aplicação web indicando que o *bot* está pronto a ser usado.

Após este processo estar concluído é preciso fazer a configuração e associação aos canais de comunicação a ser usados. Para esse efeito deve-se fazer a configuração no portal do Azure acedendo à secção de serviços de *bot* (Bot Services) e escolhendo o tipo de registo de canais (Bot Channels Registration) (Figura 30 - Criação do registo de canais de *bot*).

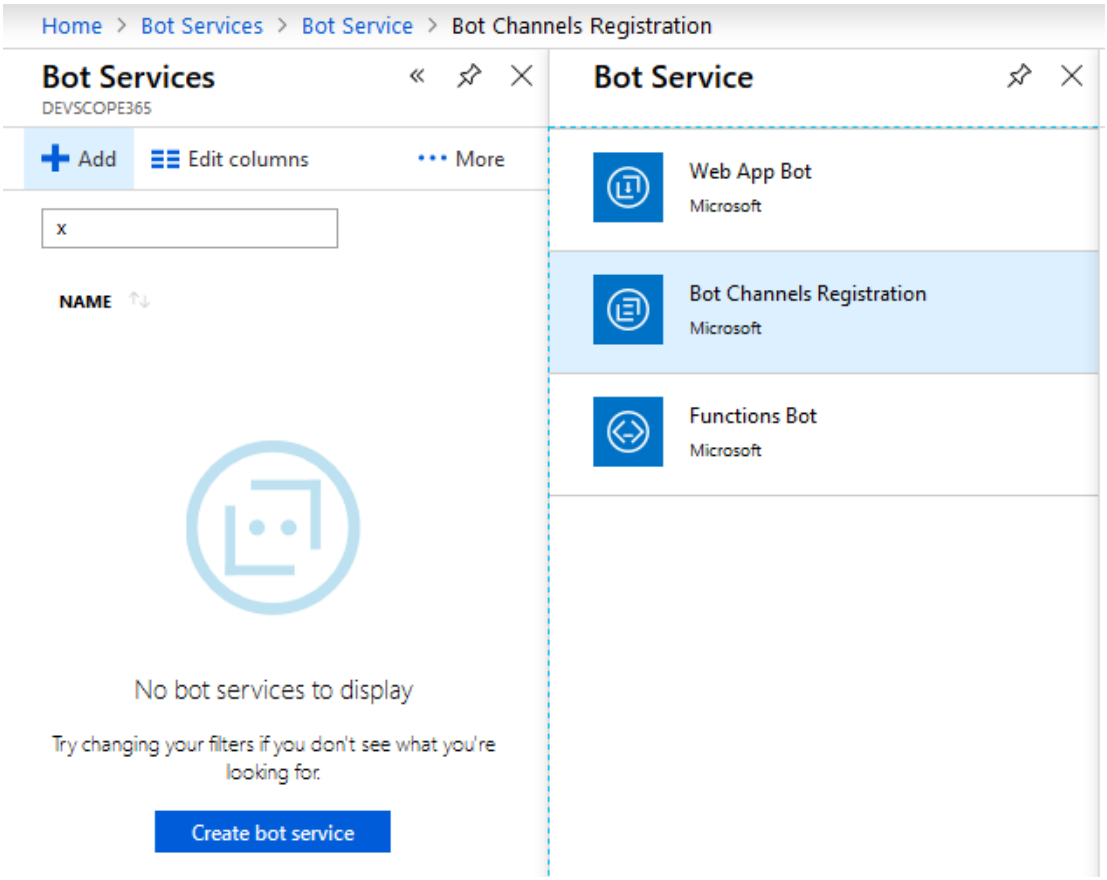

Figura 30 - Criação do registo de canais de *bot*

Este registo é usado para fazer a instanciação do serviço que gere o fluxo de conversas e ligação com os canais de comunicação (Figura 31 – Configuração de canais de comunicação).

Como o primeiro canal a usar é o Microsoft Teams, não é preciso configurar para poder ativar e usar.

No entanto, para outros canais, a configuração necessária pode ser feita acedendo à página de configuração dos mesmos.

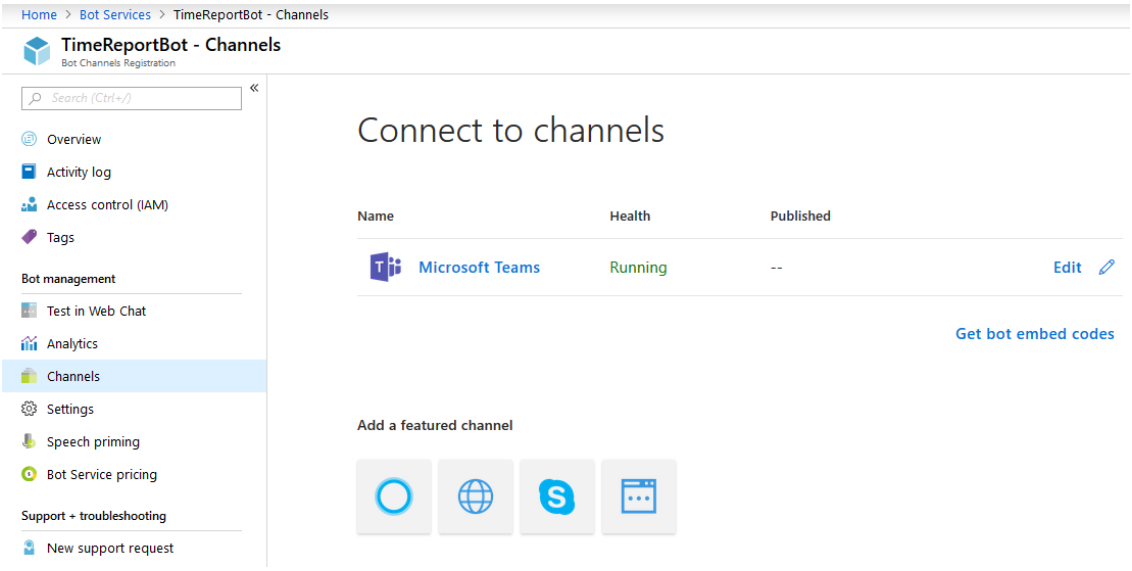

Figura 31 – Configuração de canais de comunicação

Para obter o endereço que é para distribuir pelos colaboradores pode-se carregar em "Get bot embed codes", copiar o código *HTML* fornecido (Figura 32 – Código *HTML* do botão para adicionar assistente pessoal à lista de contactos) e colar na página inicial criada na aplicação *web* onde a lógica do *bot* está alojada. Desta forma é possível partilhar esse endereço internamente e cada colaborador pode apenas usar esse botão para adicionar o assistente pessoal à lista de contactos.

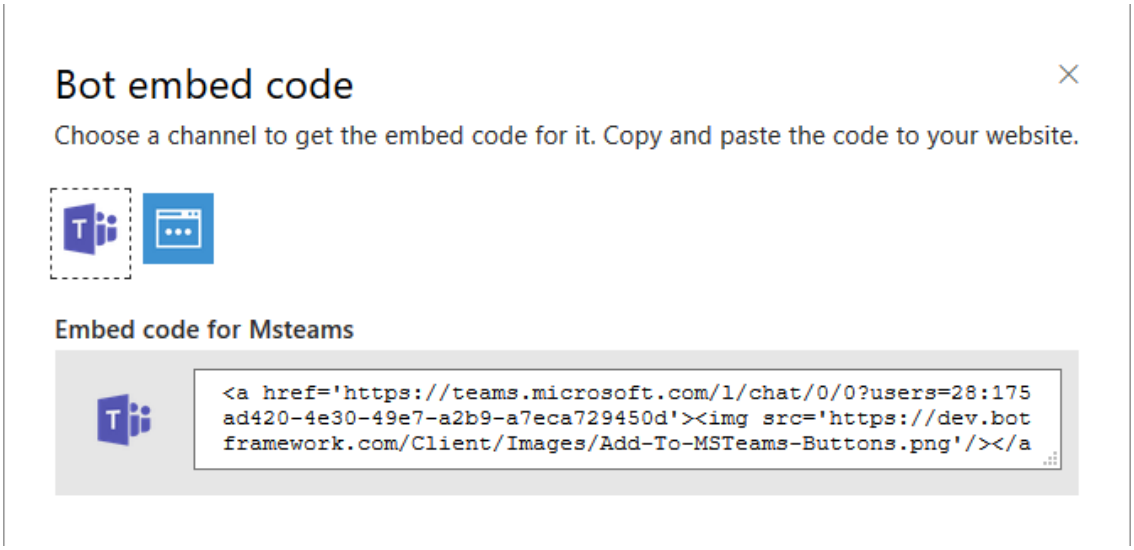

Figura 32 – Código *HTML* do botão para adicionar assistente pessoal à lista de contactos

Por último é preciso obter o registo o *bot* usando o *Bot Handle*, *App ID* e Password.

Para obter o *Bot handle* e a *App ID* pode-se aceder à secção de *Settings* (Figura 33 – *Settings* do registo de canais do *bot*).

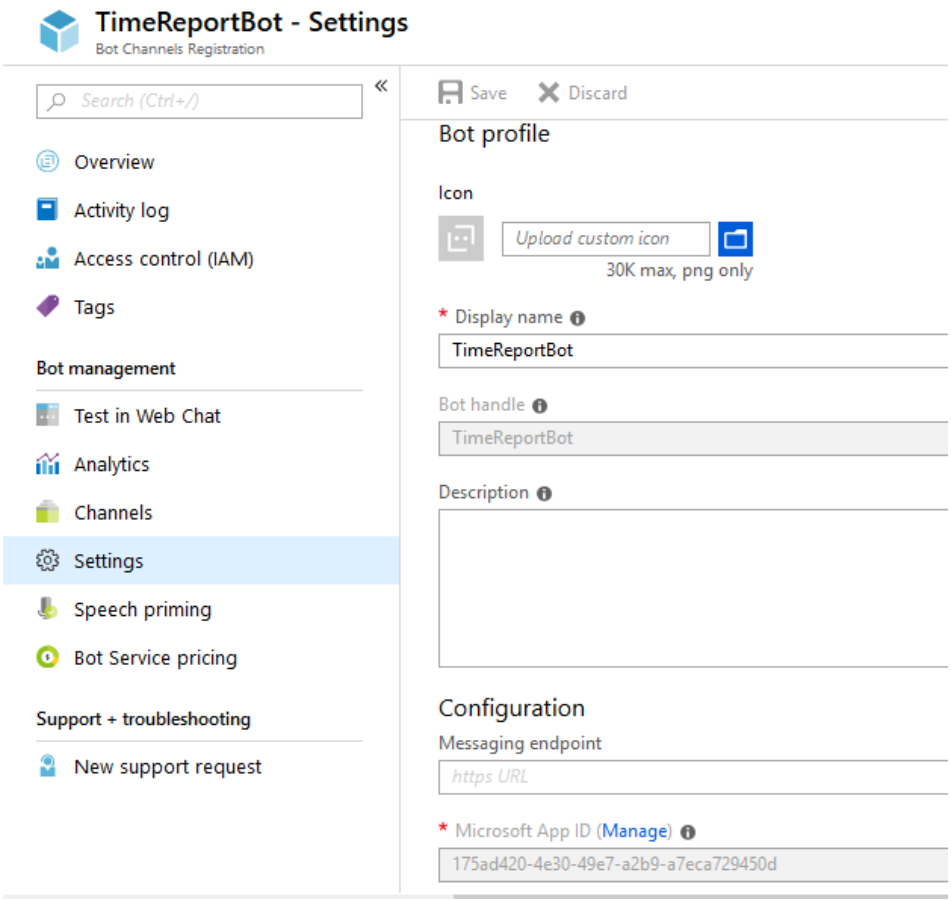

Figura 33 – *Settings* do registo de canais do *bot*

Para poder obter a password é preciso carregar em "Manage" na parte relativa à Microsoft App Id, onde é feito o reencaminhamento para o portal de registo de aplicações.

Nesse portal, após selecionar a aplicação respetiva ao equivalente App Id pode-se ver na secção das *passwords* a possibilidade de gerir as mesmas (Figura 34 – Registo de aplicação com acesso à password). Para obter uma *password* válida pode-se gerar uma nova *password* que só será visível nesta visita a esta página, pelo que deve ser guardada de imediato. Caso se perca terá de se voltar a este passo para regerar outra *password*. Por questões de segurança é aconselhável ter apenas uma em vigor.

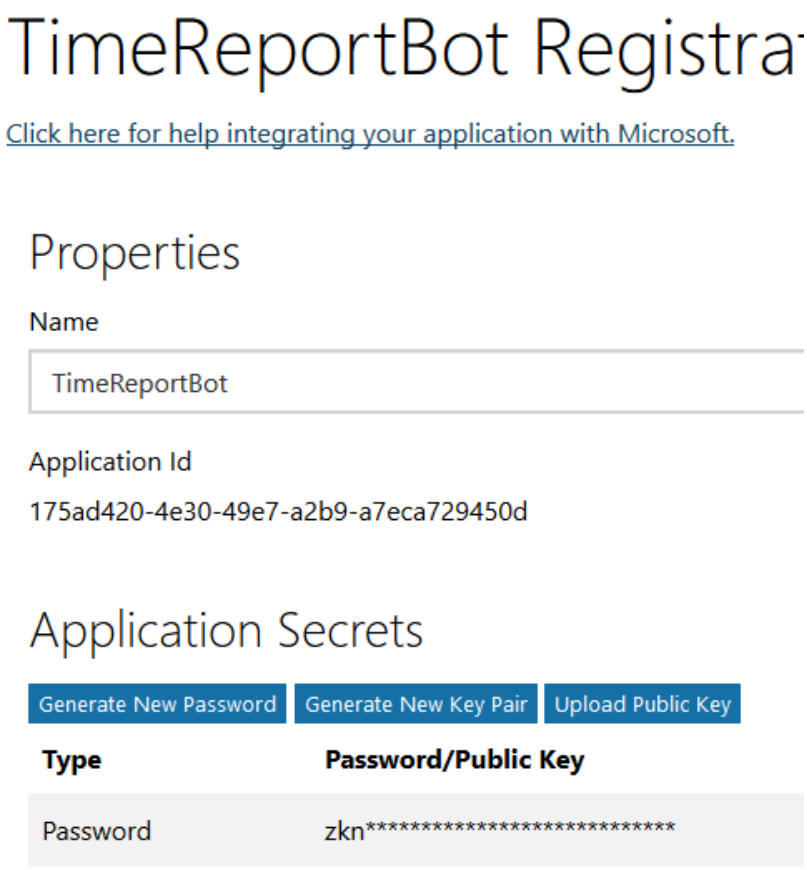

Figura 34 – Registo de aplicação com acesso à password

Esta informação deve estar incluída na configuração do *bot* na aplicação *web*.

4.5 Testes

Esta componente engloba testes unitários, funcionais e de integração onde envolve a publicação do *bot* no Azure e a associação aos canais de comunicação para a construção de cenários completos de utilização.

4.5.1 Testes unitários

No *Visual Studio* foi usado o formato da *framework* de testes criada pela Microsoft denominada por *MSTest* V2. Uma grande evolução em relação às versões anteriores foi o facto de uniformizar o suporte para o tipo de soluções. Nesta nova versão é possível usar apenas o *namespace Microsoft.VisualStudio.TestTools.UnitTesting* para obter todo o suporte necessário ao desenvolvimento de testes unitários.

Para esta solução foram criados testes unitários à camada de serviços responsáveis pelas operações ao CRM. São identificados os métodos (Figura 35 – Exemplo de testes unitários à camada de serviços) que as interfaces expõem e cria-se testes simples com o objetivo de confirmar se o *output* obtido para parâmetros de *input* conhecidos são os previsivelmente corretos.

```
[TestMethod()]
0 references | Rui Silva, 2 days ago | 1 author, 1 change
public void RetrieveUserTest()
{
    try
    {
        ICRMService crmService = new CRMService();

        var result1 = crmService.RetrieveUser("1dbb7a69-18bc-e411-80d1-00155d01d052").Result;

        Assert.IsTrue(result1.Ownerid == "1dbb7a69-18bc-e411-80d1-00155d01d052");
    }
    catch (Exception ex)
    {
        Assert.Fail(ex.Message);
    }
}

[TestMethod()]
0 references | Rui Silva, 2 days ago | 1 author, 1 change
public void SearchProjectsTest()
{
    try
    {
        ICRMService crmService = new CRMService();

        var projects = crmService.RetrieveProjects("Smartdocumentor").Result;

        Assert.IsTrue(projects.Count() > 0);
    }
    catch (Exception ex)
    {
        Assert.Fail(ex.Message);
    }
}
```

Figura 35 – Exemplo de testes unitários à camada de serviços

Um outro conjunto de testes importante de realizar é dos métodos que controlam o fluxo de lógica do *bot* para cada intenção reconhecida pelo LUIS.

Para esse efeito, são realizados testes a cada método que represente uma intenção passível de ser executada consoante a resposta vinda do serviço LUIS: *None*, *ListTimeReport* e *CreateTimeReport* (Figura 36 – Testes unitários executados aos diálogos com sucessoFigura 36 – Testes unitários executados aos diálogos com sucesso).

É de destacar que no método equivalente à intenção *CreateTimeReport*, são realizadas todas as operações até ao ponto do envio da entidade para o CRM, incluindo a obtenção de informação para a instanciação correta do objeto *TimeReportModel*.

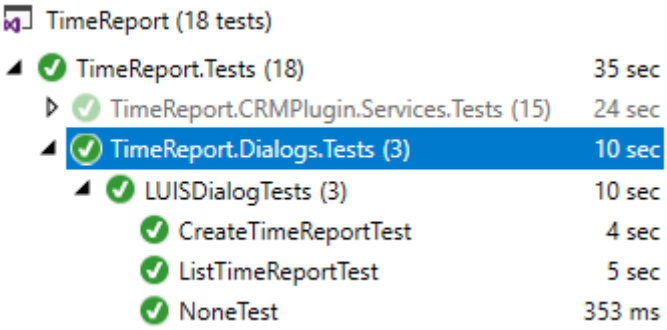

Figura 36 – Testes unitários executados aos diálogos com sucesso

4.5.2 Testes funcionais

Como o projeto no Visual Studio expõe uma *Web API* para consumir o *bot*, é possível executar localmente e testar com recurso ao Bot Framework Emulator (Figura 37 – Testes funcionais no Bot Framework Emulator), aplicação de testes desenvolvido pela Microsoft com ligação ao *endpoint* localmente criado para testes, é apresentada uma interface simples e semelhante a um canal de comunicação onde é possível experimentar a interação com o *bot*.

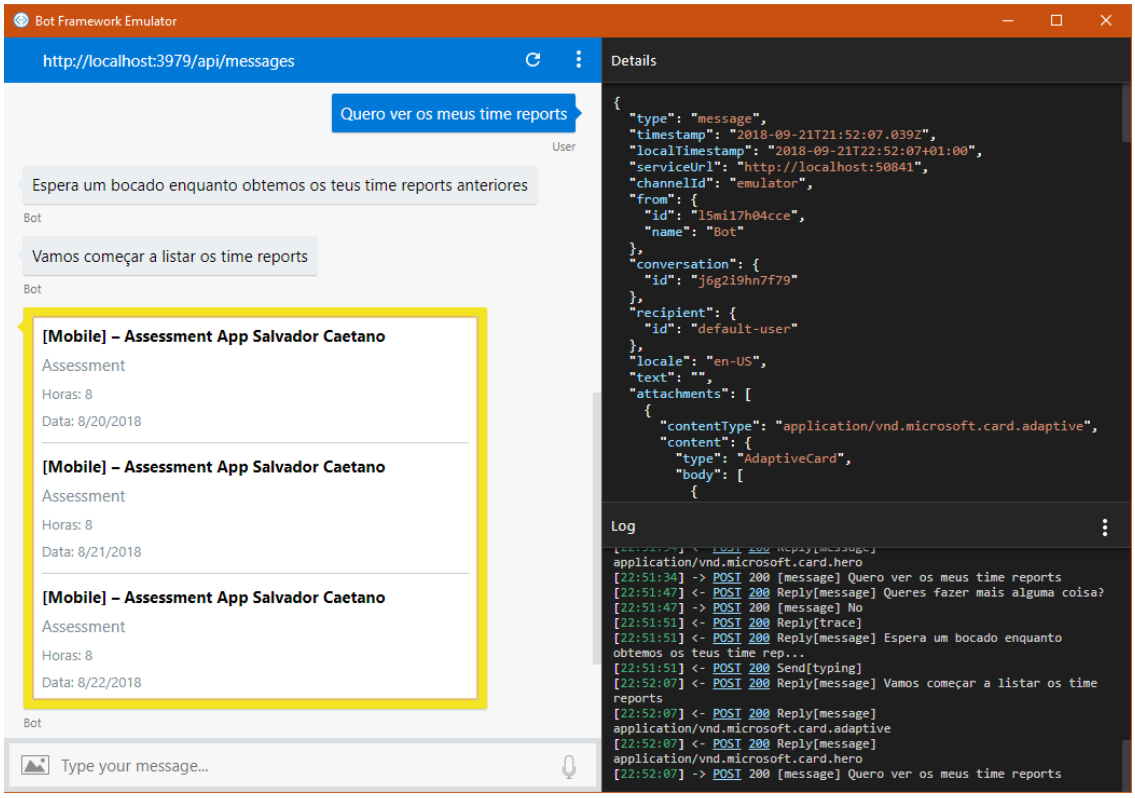

Figura 37 – Testes funcionais no Bot Framework Emulator

4.5.3 Testes de integração

São realizados testes de integração onde se pode testar todos os módulos envolvidos e ter uma perceção do funcionamento geral da solução. As funcionalidades principais que têm início a partir da iniciativa do colaborador são despoletadas através do reconhecimento da intenção do serviço LUIS e consequente reencaminhamento para o fluxo correspondente. Dessa forma pode-se isolar os componentes que se pretende testar. Esta abordagem incremental de etapas de teste assegura que seja mais fácil garantir que cada funcionalidade e componente corresponde como devido antes de se avançar para funcionalidades mais complexas (Lourenço, 2010).

Para obter um conjunto de testes funcionais mais alargado e acessível a um grupo de colaboradores, é necessário registar o *bot* no portal respetivo ao Bot Framework (Build a great conversationalist, 2017).

Através deste registo é possível associar de forma totalmente automática a adaptadores que permite o *bot* funcionar com os diferentes canais de comunicação. Acedendo a site do portal do *Bot Framework* autenticado com uma conta Microsoft, acede-se à listagem dos *bots* registados onde se tem acesso ao início do registo.

No portal do *Azure* é possível criar um *bot* com um ambiente de desenvolvimento próprio onde se pode desenvolver no browser sem necessidade de configurar uma máquina de desenvolvimento criando o que é denominado como *Functions Bot*. Desta forma, o processo de registo é mais direto com uma integração mais fácil e rápida com a plataforma que gere o registo de *bots*. No entanto, a flexibilidade e customização requerida para a solução apresentada torna esta opção algo limitativa porque o conjunto de funcionalidades e opções de integração com serviços exteriores é comparativamente menor.

Como a solução pressupõe a utilização do Visual Studio como ambiente de desenvolvimento previamente construído, é optado pelo registo de um *bot* existente usando a opção correspondente denominada como *Web App Bot* (Figura 38 – Registo de Web App *Bot*). De uma forma semelhante á criação de um *website* é apresentado um conjunto de informações necessárias dando destaque ao nome, ao plano de preços e ao tipo de bot que irá ser alojado neste registo.

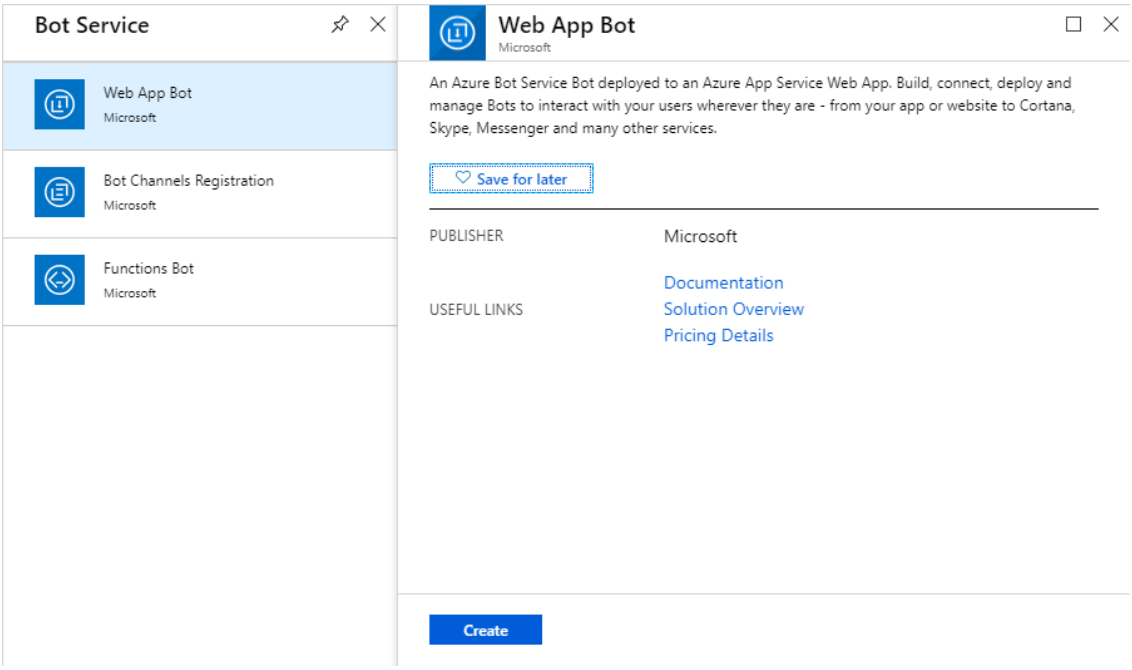

Figura 38 – Registo de Web App *Bot*

O plano de preços, ou *princing tier*, compreende dois planos que correspondem ao gratuito F0 que contém dez mil trocas de mensagens, ferramentas de construção e alguns canais de comunicação. O segundo plano é pago por conjuntos de mil trocas de mensagens, trazendo a vantagem de permitir um SLA de 99.9%. Para a presente solução, e tendo em conta a dimensão e utilização provável da mesma pela organização, é estimado que a versão gratuita F0 sirva para o que é proposto.

O tipo de *bot* é escolhido com base nos requisitos e funcionalidades principais que o *bot* terá de ter. Visto que está previsto o mesmo ser capaz de iniciar uma conversa num cenário que envolva ao final do dia relembrar o colaborador que é necessário realizar o registo de horas para o presente dia, é então necessário ser do tipo proativo (Figura 39 – Criação do ambiente azure para o *bot*).

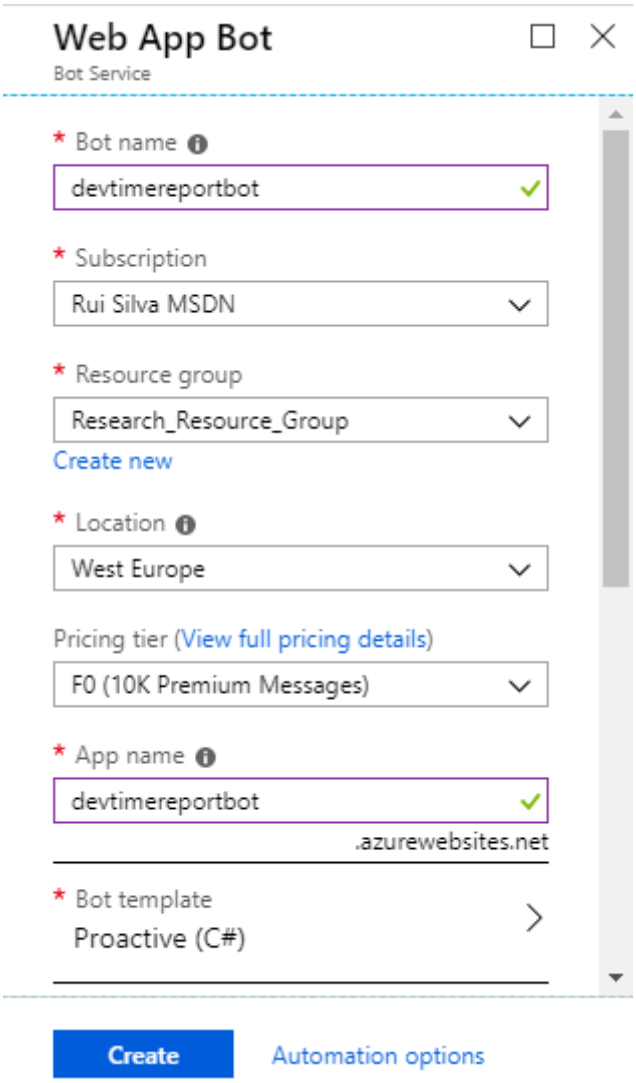

Figura 39 – Criação do ambiente azure para o *bot*

Com a instância de *Azure* criada para o ambiente de qualidade, já é possível gerar um perfil de publicação que é então usado no *Visual Studio* para publicar a solução criada. Desta forma, o isolamento de operação deste ambiente é garantido e não irá interferir mais tarde quando houver necessidade de realizar testes de qualidade de uma nova versão do *bot*.

4.5.4 Canais de comunicação

A próxima fase é dedicada à associação e configuração dos canais pelos quais vai ser possível interagir e testar.

Primeiramente é usado o Microsoft Teams. Suporta o tipo de autenticação que se procura para obter as credenciais dos colaboradores.

Numa fase posterior é adicionado o suporte para o Skype e Slack. Outros canais foram ponderados pela administração, mas numa última análise notou-se que a configuração necessária tinha uma complexidade que não seria benéfica para o âmbito deste trabalho.

5 Avaliação da solução

É criado um grupo de controlo que usa o método atual criando o registo de horas de forma totalmente manual no portal do CRM (*Costumer Relationship Manager*) e um outro grupo experimental que irá usar *bot* como novo método, totalmente independente de interações com o CRM, para realizar o mesmo registo. Cada grupo é composto por 12 pessoas qua abrangem todas as áreas de operação da organização.

Pretende-se desenvolver métodos de avaliação que tentem comprovar que a utilização de um *bot* desenvolvido para agilizar o processo de registo de *time reports* possa ser em ambiente corporativo com benefício face ao processo atualmente em vigor.

É medida a facilidade com que se efetua um registo de *time report* com recurso ao *bot* e estabelecer um grau de comparação ao método atual.

Foi efetuado um questionário (ver Anexo 2 – Questionário de avaliação compreensiva da solução) que visa a entender o panorama das interações com o portal de CRM e com o assistente pessoal virtual desenvolvido nesta solução com base no grupo de controlo e grupo experimental, respetivamente.

São precisas duas perguntas para uma avaliação compreensiva da solução:

- Em média, com que regularidade são registados os *time reports*?
- Em média, quanto tempo demora a registar os *time reports*?

A distribuição do questionário coincide com o fim de duas semanas após o grupo experimental começar a utilizar exclusivamente o assistente pessoal. Todos os elementos de ambos os grupos cumpriram o preenchimento.

Feita a análise dos resultados do grupo de controlo (Figura 40 – Resultados de tempo médio de registo via portal do CRM) repara-se numa tendência crescente no tempo que é dispensado para o preenchimento dos mesmos à medida que a periodicidade com que se realiza os registos aumenta.

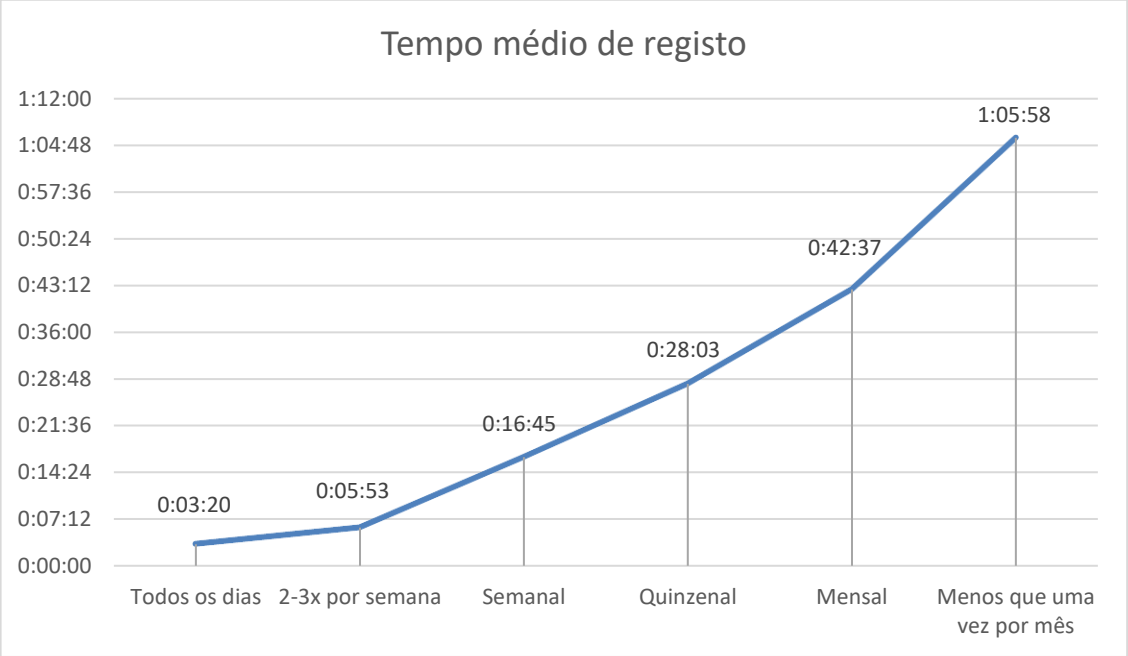

Figura 40 – Resultados de tempo médio de registo via portal do CRM

No entanto, no grupo experimental já é possível verificar que o tempo médio para o registo é menor (Figura 41 – Resultados do tempo médio de registo via assistente pessoal virtual). Estes resultados devem-se principalmente ao conjunto de informações de contexto que é possível adiantar para cada registo tornando toda a experiência mais simples e rápida de processar pelo colaborador.

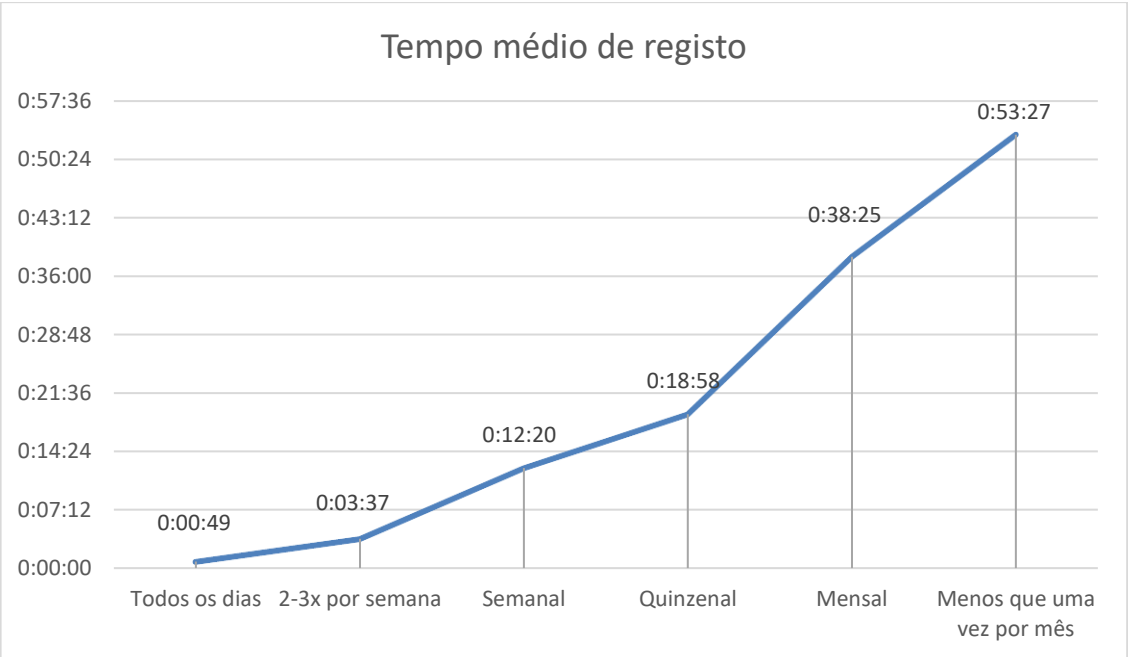

Figura 41 – Resultados do tempo médio de registo via assistente pessoal virtual

Um dos problemas principais que a solução desenvolvida visa a tentar resolver é a falta de detalhe devido ao esquecimento dos registos de *time reports*.

Algumas equipas da organização começaram a organizarem-se com base em lembretes no calendário. No entanto, nem sempre é oportuno reagir nesses momentos e acabam por ser esquecidos novamente.

Os resultados à segunda pergunta (Figura 42 – Periodicidade do registo de *time reports* por parte dos colaboradores usando o portal CRM) demonstram que uma grande parte dos colaboradores apenas realizam os registos menos duas a três vezes por semana, chegando a casos preocupantes que envolvem atrasos de mais de um mês.

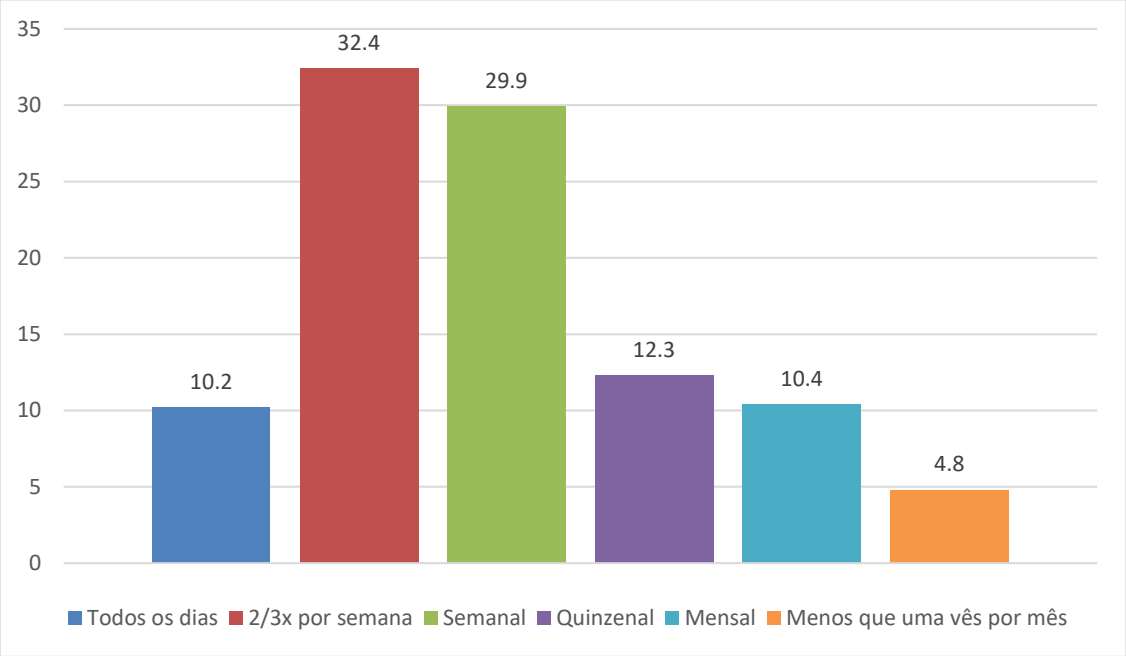

Figura 42 – Periodicidade do registo de *time reports* por parte dos colaboradores usando o portal CRM

Já o grupo experimental, os resultados (Figura 43 - Periodicidade do registo de *time reports* por parte dos colaboradores usando o assistente pessoal virtual) demonstram que com recurso a lembretes mais interativos e com seguimento para a sua resolução imediata, demonstra uma clara melhoria na regularidade dos registos colmatando cenários onde é fácil perder detalhe por falta de memória.

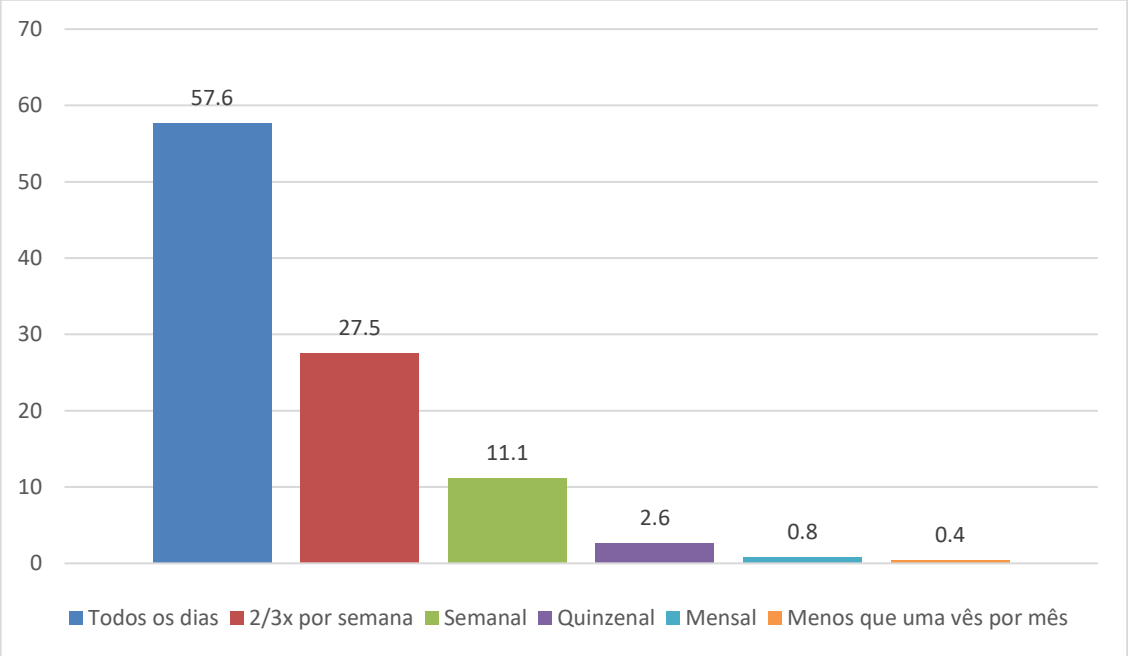

Figura 43 - Periodicidade do registo de *time reports* por parte dos colaboradores usando o assistente pessoal virtual

Relativamente à prestação do serviço LUIS, a medição da eficácia a reconhecer a intenção dos colaboradores é feita com base na distribuição dos resultados que o são retornados (Figura 44 – Distribuição das intenções reconhecidas pelo LUIS).

No momento de recolha dos resultados havia 231 interações com a app LUIS.

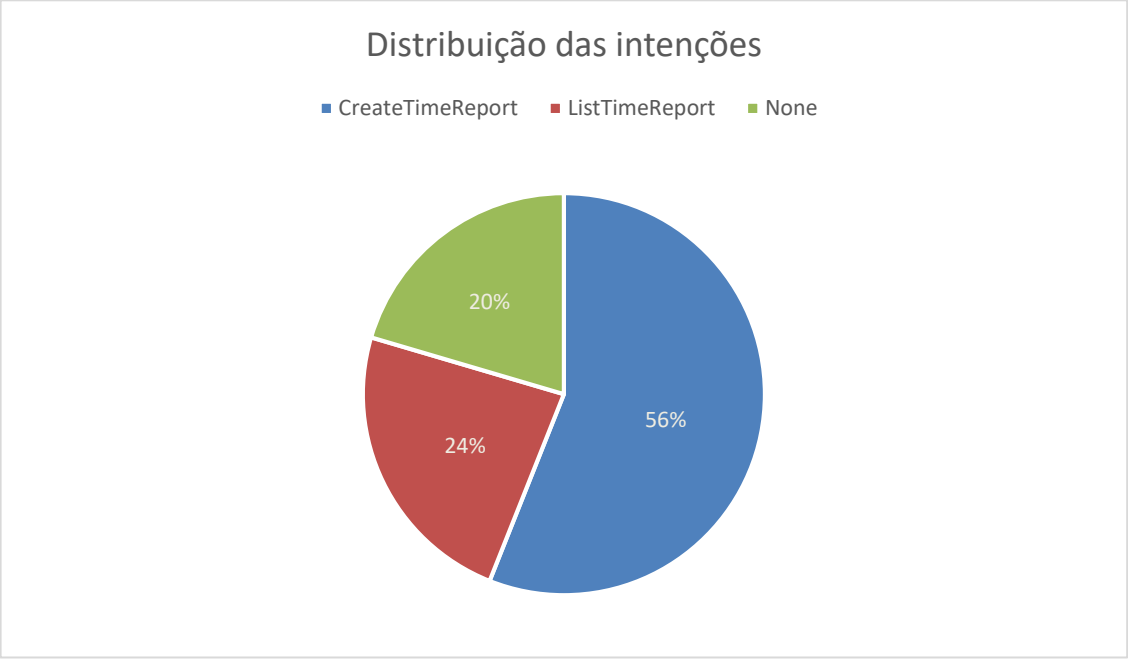

Figura 44 – Distribuição das intenções reconhecidas pelo LUIS

Pode-se reparar que a terceira intenção mais vezes reconhecida pelo modelo do LUIS é o "None". Estes resultados demonstram uma de duas hipóteses:

- Num total de 20% dos *inputs* fornecidos para classificação, não houve certeza para reconhecer qualquer uma das outras intenções. Para resolver o problema, deve-se rever esses *inputs* e classificar na intenção que mais se encaixa nos casos que fizer sentido;
- Os *inputs* tratam-se deliberadamente de intenções fora do domínio e não pertencem a qualquer outra intenção. Nestes casos não é necessário tomar qualquer ação.

Em relação às entidades reconhecidas pode-se concluir que os colaboradores usaram frases que incluem entidades que visam a acelerar o processo de preenchimento (Figura 45 – Distribuição das entidades reconhecidas do LUIS).

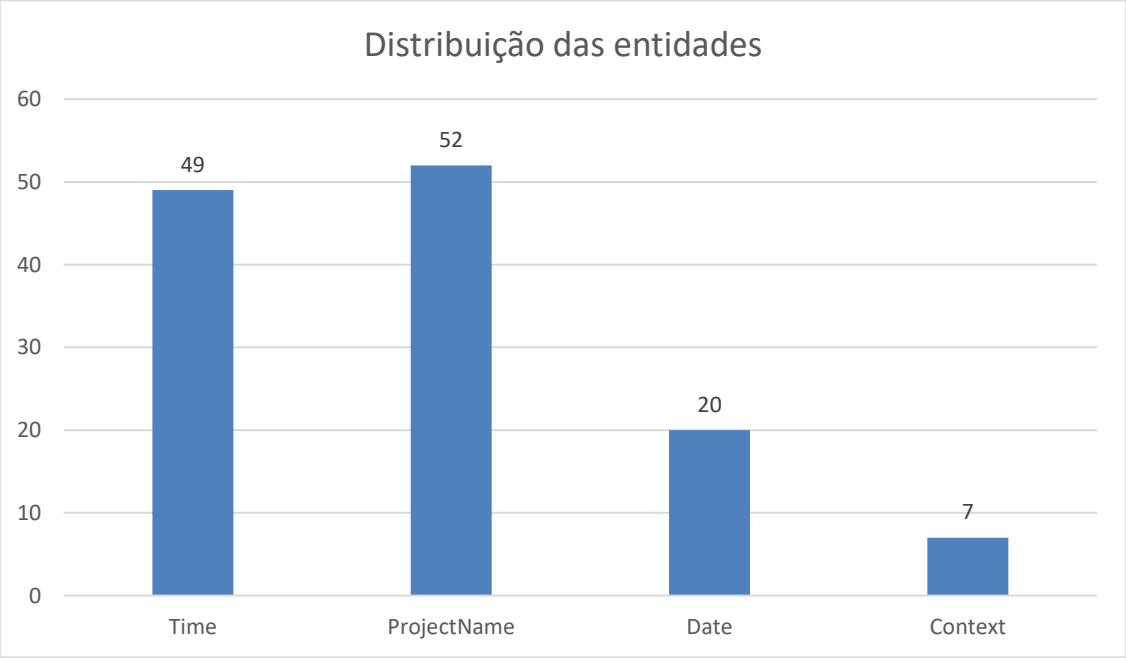

Figura 45 – Distribuição das entidades reconhecidas do LUIS

6 Conclusão

Aquando da geração da ideia, a noção que se obteve foi que a tecnologia estaria ainda em fases muito iniciais, e que isso representasse uma dificuldade acrescida na especificação e desenvolvimento da solução. No entanto, no decorrer da construção e implementação da solução, foi possível verificar um avanço considerável na tecnologia. Esse avanço traduziu-se em diversas alterações de forma a melhorar a qualidade da solução final.

O ciclo de desenvolvimento e melhoria do tipo de solução de software implementado difere consideravelmente do que se espera de uma aplicação de software ou até mesmo de uma solução *web*. Existe uma considerável diferença na mentalidade necessária para implementar uma experiência de utilizador adequada para se enquadrar numa conversa, onde muitas vezes não é possível dar o contexto completo de forma simples ao colaborador. (DeWael, 2017)

Os principais benefícios na utilização de assistentes pessoais são os seguintes:

- Parece mais natural que a utilização de aplicações. Os utilizadores interagem usando comandos de texto em linguagem natural;
- Não há necessidade de aprender novas competências para poder interagir com os assistentes pessoais, desde que estes sejam bem implementados;
- Há uma maior possibilidade de preparar um assistente a adaptar-se ao utilizador;
- Não há necessidade de instalar software adicional ou estar limitado no acesso ao assistente.

Os cenários mais indicados para a utilização dos assistentes pessoais virtuais centram-se no consumo de informação, oportunidades comerciais como angariação de novos e suporte. (Babich, 2018)

Atualmente, as interfaces gráficas em ambientes profissionais tendem a ser complicadas e cheias de opções. Os colaboradores sentem-se habituados a esta tendência. Numa conversa com um assistente virtual há um espaço limitado para expor todas as opções e, por isso, deve-se subdividir e organizar as opções em componentes. Cada componente deve ser encaminhado para as opções que apenas existem no contexto presente. O fluxo de conversa de cada componente tende a ser limitado apenas ao contexto e domínio ao qual esse componente se

compromete a responder. A divisão das funcionalidades expostas pela solução desenvolvida foi resultado da análise dos diferentes casos de uso.

O entendimento do estado da arte foi crucial para entender qual o papel a desempenhar por um assistente pessoal virtual, nomeadamente que tipo de problemas é capaz de resolver. A analise dos tipos de *bots* que existiram no passado em comparação aos que existem predominantemente nos dias atuais permitiu enquadrar o contexto e domínio da solução presente neste documento.

Existe um conjunto alargado de ferramentas que facilitam a criação e desenvolvimento de *bots* e assistentes pessoais virtuais inteligentes usando processamento em linguagem natural com recurso ao *machine learning*.

Muitas dessas ferramentas são comerciais e proprietárias das organizações que as criaram tornando o publico alvo mais limitado. Exemplos dessa prática podem ser observados pelas soluções providenciadas pelo Facebook e Amazon.

A Microsoft tem uma abordagem diferente na forma como democratiza as suas ferramentas. Torna a sua utilização mais flexível com um maior número de integrações e um modelo de trabalho mais abrangente. Em contrapartida, para utilizar as ferramentas da Microsoft é necessário consumir a plataforma Azure. Esta plataforma é completa e fornece todas as peças necessárias para um desenvolvimento sem grande dificuldade.

De forma a optar pelo melhor conjunto de ferramentas a usar para o desenvolvimento da solução foi tido em conta o histórico da relação que a organização tem com a Microsoft e por sua vez na utilização dos seus serviços.

Apesar da decisão tomada, é preciso salientar que tem havido um crescente potencial nas ferramentas de criação de *bots open source* onde se verifica uma contribuição da comunidade.

Ao desenhar a estrutura da solução foram utilizados padrões de programação orientada a objetos que garantem uma maior flexibilidade recorrendo a uma maior separação do código em diversos conjuntos de responsabilidades.

Foi usado o padrão de adaptadores (Adapter Pattern, 2014), onde originalmente há uma interface intermédia que faz a conversão do que é esperado para a comunicação com a API

OData, anteriormente desenvolvida para a comunicação com o CRM (*Costumer Relationship Manager*), e a lógica do *bot*.

As conversas seguem árvores de fluxos que são correspondidas por componentes. Desta forma é usado o padrão de cadeia de responsabilidades (Chain Of Responsibility Pattern, 2014). No final da execução de cada fluxo, é retornado o resultado ao componente anterior até chegar ao início de uma *stack*. Nessa fase é usada uma frase que indica ao colaborador que a conversa terminou e que o assistente está disponível para próximas interações.

Na avaliação da solução, ficou claro que a velocidade e praticidade com que se realizam os registos foi substancialmente melhorada na comparação entre o grupo de controlo experimental. Ao aceder ao portal do CRM, o colaborador via-se obrigado a concentrar-se apenas no registo numa altura em que podia não se lembrar dos pormenores importantes.

A memória associativa tem um papel importante no momento ideal para um colaborador se lembrar dos detalhes que tornam um registo mais completo (Valle, 2018). Quando o ambiente proporciona as condições ideais é preciso haver uma forma prática e breve para aproveitar a oportunidade.

Ao proporcionar uma solução que se adapta ao ritmo do colaborador as vantagens surgem em forma de resultados mais precisos e uma satisfação geral maior.

Por fim, este trabalho serve de referência no sentido de melhorar os métodos de organização de cada colaborador e a qualidade de reportagem dos registos de horas que são cruciais à organização.

6.1 Desenvolvimentos futuros

Apesar do âmbito do trabalho realizado estar limitado à criação e listagem de time reports, foram exploradas ideias sobre possíveis melhorias.

Os pontos identificados são os seguintes:

- Dotar o assistente pessoal da capacidade de usar a síntese de voz para falar com o colaborador onde se aplica com os conjuntos de frases apropriadas a cada ação que o

mesmo já apresenta. Apenas quando o colaborador usar a voz para indicar input é que deve haver resposta pelo mesmo meio de volta;

- Introdução de intenções mais variadas no LUIS para complementar a parte conversacional do assistente pessoal virtual com exemplos de interação tais como "greetings", "bye", "thank", "swear" e “repeatLast”;
- Introdução de sistema de feedback após completude de ações usando o *bot*. O *feedback* é guardado numa BD que pode ser consumida posteriormente por um *dashboard* em Microsoft PowerBI para poder analisar estatisticamente o comportamento e desempenho do assistente pessoal virtual. No final de cada diálogo principal (CreateTimeReport e ListTimeReports) pode haver oportunidade de perguntar, de forma aleatória, qual foi a experiência do colaborador na utilização do *bot*. O feedback pode ser composto pela seleção de níveis de satisfação (1 a 10) e possibilidade de escrever um comentário. Se o colaborador fornecer comentários , pode-se recorrer a um analisador de sentimento, aplicando modelos de machine learning treinados para o efeito e guardar os resultados em conjunto com o feedback;
- Assistente pessoal virtual de registo de despesas;
    - Depois de registar os time reports deve poder listar os mesmos para selecionar;
    - Depois de selecionar o colaborador deve dar as informações necessárias para efetuar o registo de despesa pedindo o valor, categoria e descrição. Adicionalmente deve poder fazer upload de uma imagem que represente a fotografia ou digitalização da fatura que posteriormente é submetida para o Smartdocumentor (produto desenvolvido pela organização para gestão documental com reconhecimento de OCR) para ser usada para efetuar treino ao modelo de reconhecimento de dados. Pode ser preciso uma Review Station (componente integrante do produto) para fazer verificações manuais por parte dos colaboradores responsáveis por confirmar as despesas dos colaboradores;
    - No final da recolha de informação necessária é mostrado um resumo da despesa prestes a registar e associar ao time report.
- O projeto pode evoluir no sentido de suportar plugins para outros providers de serviços de registo de horas para além do CRM (Costumer Relationship Manager) Microsoft Dynamics 365. É um desafio de engenharia de software relevante e um caso de estudo de exemplo.

Como nota final, o desenvolvimento deste trabalho permitiu abrir um conjunto de possibilidades que envolvem um âmbito mais alargado e completo. Da forma como este foi desenvolvido permite que o melhorias e entregas continuas sem disrupção da utilização do mesmo.

7 Referências

A flexible & easy-to-manage web server. (2018). Obtido em 2 de Abril de 2017, de Microsoft IIS: https://www.iis.net/

Adapter Pattern. (14 de Outubro de 2014). Obtido de WikiWikiWeb: http://wiki.c2.com/?AdapterPattern

Amazon. (s.d.). *Amazon Polly.* Obtido em 8 de 10 de 2017, de AWS: https://aws.amazon.com/polly/

Babich, N. (13 de Fevereiro de 2018). *UI of the Future: Conversational Interfaces.* Obtido de Shopify Partners: https://www.shopify.com/partners/blog/conversational-interfaces

Brandl, K., Standefer, R., Vo, D. C., Dempsey, J., & Delimarsky, D. (21 de Junho de 2017). *Basic features of FormFlow.* Obtido em 10 de Setembro de 2017, de Microsoft Docs: https://docs.microsoft.com/en-us/bot-framework/dotnet/bot-builder-dotnet-formflow

Brown, B. (15 de Dezembro de 2015). *Botkit - Building Blocks for Building Bots.* Obtido de Github: https://github.com/howdyai/botkit

Build a great conversationalist. (7 de Janeiro de 2017). Obtido de Microsoft Bot Framework: https://dev.botframework.com/

Chain Of Responsibility Pattern. (8 de Julho de 2014). Obtido de WikiWikiWeb: http://wiki.c2.com/?ChainOfResponsibilityPattern

Chatbot search trend. (17 de Setembro de 2017). Obtido de Google Trends: https://trends.google.pt/trends/explore?date=today%205-y&q=chatbot

Chatbot vs Traditional Service Apps. (12 de Julho de 2017). Obtido em 10 de Outubro de 2017, de Chatbot's Life: https://chatbotslife.com/chatbot-replacing-the-apps-e0095feb030d

Cognitive Services. (16 de Setembro de 2017). Obtido de Microsoft Azure: https://azure.microsoft.com/en-us/services/cognitive-services/

Crie bots de conversação. (s.d.). Obtido em 14 de Janeiro de 2017, de Amazon Lex: https://aws.amazon.com/pt/lex/

Delgado, A. S. (1997). *Agentes de Software: Conceitos e Tecnologias.* Obtido em 20 de Outubro de 2017, de INESC - Instituto de Engenharia de Sistemas e Computadores: http://isg.inesc-id.pt/alb/static/papers/1997/n6-as-encei-OE-1997.pdf

DeWael, S. (21 de Junho de 2017). *Finding your way around the world of bots as a web developer.* Obtido de Slack Platform Blog: https://medium.com/slack-developer-blog/finding-your-way-around-in-the-world-of-bots-1bbf9cece6fb

Dynamics 365. (2017). Obtido em 6 de Junho de 2017, de Microsoft: https://www.microsoft.com/en-us/dynamics365/first-look

Finjan Mobile. (2017 de Maio de 9). *App Permissions – The Good, The Bad, and Why You Need to Pay Attention*. Obtido de Finjan Mobile: https://www.finjanmobile.com/app-permissions-the-good-the-bad-and-why-you-need-to-pay-attention/

Gestão de API. (3 de fevereiro de 2018). Obtido de Microsoft Azure: https://azure.microsoft.com/pt-pt/services/api-management/

Gorgens, E. B. (Dezembro de 2009). *Estimação do volume de árvores utilizando redes neurais artificiais.* Obtido em 12 de Julho de 2017, de Revista Árvore: http://dx.doi.org/10.1590/S0100-67622009000600016

Howdyai. (25 de Fevereiro de 2017). *Botkit.* Obtido em 16 de Março de 2017, de Github: https://github.com/howdyai/botkit

Internet bot. (s.d.). Obtido em 3 de 3 de 2017, de Wikipédia: A Enciclopédia Livre: http://en.wikipedia.org/wiki/Internet_bot

Jones, L. (1977). Employee motivation through job enrichment. *Journal of Education and Training, 19*(7), 214-219. Obtido em 24 de 8 de 2018, de http://emeraldinsight.com/doi/abs/10.1108/eb001962

Knecht, T. (21 de Setembro de 2016). *A Brief History Of Bots And How They've Shaped The Internet Today*. Obtido de Abusix: https://www.abusix.com/blog/a-brief-history-of-bots-and-how-theyve-shaped-the-internet-today

Kongthon et al, A. (27 de october de 2009). *Implementing an online help desk system based on conversational agent.* Obtido de ACM DL: https://dl.acm.org/citation.cfm?id=1643823.1643908

Lippert, E. (15 de Outubro de 2012). *Is C# a strongly typed or a weakly typed language?* Obtido de Microsoft Developer:

https://blogs.msdn.microsoft.com/ericlippert/2012/10/15/is-c-a-strongly-typed-or-a-weakly-typed-language/

Lourenço, M. (Janeiro de 2010). *Teste de Integração.* Obtido de Qualidade de Software : http://qualidade-de-software.blogspot.com/2010/01/teste-de-integracao.html

Murgia, M. (13 de Abril de 2016). *Facebook Messenger's new bots are a powerful way to target adverts.* Obtido em 20 de Março de 2017, de The Telegraph: http://www.telegraph.co.uk/technology/2016/04/12/facebook-messenger-launches-chat-bot-economy-to-take-on-apps/

Nandwani, K. (10 de Janeiro de 2018). *An introduction to NuGet*. Obtido de Microsoft Nuget: https://docs.microsoft.com/en-us/nuget/what-is-nuget

Outlaw, C. L. (14 de Junho de 2017). *Create an ASP.NET web app in Azure.* Obtido em 11 de Setembro de 2017, de Microsoft Azure Docs: https://docs.microsoft.com/en-us/azure/app-service/app-service-web-get-started-dotnet#publish-to-azure

Petrusha, R. (30 de Março de 2017). *Overview of the .NET Framework.* Obtido de Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/framework/get-started/

Rouse, M., & Botelho, B. (Outubro de 2017). *Virtual Assistant (AI assistant)*. Obtido de Techtarget: https://searchcrm.techtarget.com/definition/virtual-assistant

Sg, K., & Ja, K. (1975). Job enrichment: what it is and how to measure it. *Respiratory Care, 20*(6), 543. Obtido em 24 de 8 de 2018, de https://ncbi.nlm.nih.gov/pubmed/10314982

Standefer, R. (24 de agosto de 2018). *Manage state data*. Obtido de Microsoft Azure: https://docs.microsoft.com/en-us/azure/bot-service/dotnet/bot-builder-dotnet-state?view=azure-bot-service-3.0#in-memory-data-storage

Stott, L. (5 de Abril de 2016). *What is Microsoft Bot Framework Overview.* Obtido de Microsoft Developer: https://blogs.msdn.microsoft.com/uk_faculty_connection/2016/04/05/what-is-microsoft-bot-framework-overview/

Thomas, G. (06 de Agosto de 2016). *Chatbot ppt.* Obtido em 12 de Agosto de 2017, de SlideShare: https://pt.slideshare.net/GeffThomas/chatbot-ppt

Umstot, D. D. (1976). Effects of job enrichment and task goals on satisfaction and productivity: Implications for job design. *Journal of Applied Psychology*, 61(4), 379.

Valle, M. E. (1 de Agosto de 2018). *Os Tipos de Memória.* Obtido de Memorização: https://www.researchgate.net/publication/228896978_IFAMs-memorias_associativas_baseadas_no_aprendizado_nebuloso_implicativo

Vlad Vinogradsky. (20 de junho de 2018). *What is API Management?* Obtido de Microsoft Azure Docs: https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts

8 Anexos

8.1 Anexo 1 – Questionário de satisfação do Portal CRM

Inquérito de Satisfação

	1 – Nada fácil	2	3	4	5	6	7	8	9	10 – Muito fácil
Facilidade de operação										
Intuitivo										
Fácil de responder										
Tempo médio por registo										
Satisfação geral										

8.2 Anexo 2 – Questionário de avaliação compreensiva da solução

	Tempo em minutos e segundos
Em média, com que regularidade são registados os *time reports*?	
Em média, quanto tempo demora a registar os *time reports*?	