



UNIVERSIDADE ESTADUAL DE SANTA CRUZ – UESC

GABRIEL RODRIGUES DOS SANTOS

LANA, ASSISTENTE PESSOAL: Sistema distribuído para recuperação de informações no âmbito da UESC

**ILHÉUS - BAHIA
2018**

GABRIEL RODRIGUES DOS SANTOS

LANA, ASSISTENTE PESSOAL: Sistema distribuído para recuperação de informações no âmbito da UESC

Trabalho de Conclusão de Curso apresentado à Universidade Estadual de Santa Cruz - UESC, como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Leard de Oliveira Fernandes

GABRIEL RODRIGUES DOS SANTOS

LANA, ASSISTENTE PESSOAL: Sistema distribuído para recuperação de informações no âmbito da UESC

Trabalho de Conclusão de Curso apresentado à Universidade Estadual de Santa Cruz - UESC, como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

Ilhéus, 5 de dezembro de 2018.

Prof. Me. Leard de Oliveira Fernandes
UESC/DCET
(Orientador)

Prof. Dr. Francisco Bruno Souza Oliveira
UESC/DCET

Prof. Dr. Marcelo Ossamu Honda
UESC/DCET

À minha família e à minha companheira Nilana que, com muito amor e paciência, sempre me deram todo o suporte quando necessário durante toda a minha etapa de graduação.

AGRADECIMENTOS

À Universidade Estadual de Santa Cruz, pela oportunidade de fazer o curso.

Ao NBCGIB, pelo ambiente criativo e amigável que me proporcionou desde o início do curso.

Ao meu orientador Prof. Me. Leard de Oliveira Fernandes, pelo suporte no pouco tempo que lhe coube, pelas suas correções e incentivos.

Ao Prof. Dr. Marcelo Ossamu Honda, pelo apoio e oportunidades dadas durante todo meu percurso na universidade e pelas orientações de iniciações científicas.

Aos meus pais, Júnior e Mara, e irmãos, Vinícius e Neto, pelo amor, incentivo e apoio incondicional.

À minha companheira Nilana, pelo suporte emocional e incentivador nas horas difíceis, de desânimo e cansaço, durante toda a minha trajetória nesta universidade e pela ajuda dada na escrita e revisão deste projeto.

À minha cunhada Laylla, pela ajuda e apoio fornecidos para a escrita e revisão deste projeto.

Aos integrantes da “*semi.pro*”: Adshow, Pedreca, Medina, Bida e Pague, também aos meus amigos, em especial Alberto, Levy, Tulio e Daniel, por todos esses anos de apoio e companheirismo e pela ajuda no amadurecimento e testes deste projeto.

Aos professores e funcionários pelos ensinamentos e pela convivência durante este período.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“Stay hungry, stay foolish.”

(Steve Jobs)

LANA, ASSISTENTE PESSOAL: Sistema distribuído para recuperação de informações no âmbito da UESC

RESUMO

Com a disseminação de computadores pessoais e dispositivos móveis conectados à internet, junto ao avanço da tecnologia cognitiva na área de linguagem natural, softwares de assistência pessoal surgiram para uma variedade de propósitos. A interação entre usuário e assistente pessoal, normalmente, se dá através da troca de mensagens, onde o usuário faz uma pergunta ou uma requisição de serviço, e o assistente a responde, ou executa o serviço requisitado. O objetivo deste trabalho é implementar um *software* de assistência pessoal distribuído, nomeado como Lana, capaz de receber mensagens textuais, interpretá-las, executar serviços quando necessário, e retornar ao usuário final uma resposta trivial ou um resultado da execução de um serviço requisitado. Serviços de recuperação de informações sobre a UESC disponíveis no *site* da própria universidade e informações do portal sagres, do aluno e do professor, foram implementados nos extratores de dados. Para a implementação do *software* de assistência pessoal foi utilizado um serviço de entendimento de linguagem natural, uma solução *backend* como serviço para o armazenamento de dados dos usuários, *web scraping* para recuperar informações, serviços de hospedagem e as linguagens de programação JavaScript e Python. Por fim, foi desenvolvido um *software* distribuído que se mostrou capaz de receber mensagens textuais a partir de duas aplicações de troca de mensagens, extrair as intenções e entidades das mensagens dos usuários e recuperar informações do site da UESC e do portal acadêmico sagres quando necessário.

Palavras-chave: Assistente Pessoal. IBM Watson. Sistemas Distribuídos. Web Scraping.

LANA, PERSONAL ASSISTANT: Distributed system for information retrieval
regarding UESC

ABSTRACT

With the spread of personal computers and mobile devices connected to the internet, along with the advancement of cognitive technology in the area of natural language, personal assistance softwares has emerged for a variety of purposes. The interaction between the user and a personal assistant usually occurs through the exchange of text messages, where the user asks a question or request a service, and the assistant responds or performs the requested service. The objective of this work is to implement distributed personal assistance software, named Lana, capable of receive textual messages, interpret them, perform actions when necessary, and return to the user a trivial response or the result of a requested service. Information retrieval services about the UESC available on the university's own website and information from the portal sagres, for students and teachers, were implemented in data extractors. For the implementation of the personal assistance software, were used a natural language understanding service, a backend as a service solution for storing users' data, web scraping to retrieve information, hosting services and the JavaScript and Python programming languages. Finally, one distributed software was developed that was able to receive textual messages from two messaging applications, extract the intents and entities from the user's messages and retrieve information from the UESC website and the sagres academic portal when necessary.

Keywords: Personal Assistant. IBM Watson. Distributed Systems. Web Scraping.

LISTA DE FIGURAS

Figura 1 - Painel do serviço Watson Assistant com algumas Intents criadas para o workspace Lana.	33
Figura 2 - Painel do serviço Watson Assistant com todas as Entities criadas para o workspace Lana.	35
Figura 3 - Painel do serviço Watson Assistant com parte do fluxo de primeira conversa criada para o workspace Lana.....	36
Figura 4 - Topologia de comunicação entre módulos.	44
Figura 5 – Primeira conversação com a Lana e serviços ofertados.	45
Figura 6 - Lana informando suas principais funcionalidades.	46
Figura 7 - Lana realizando ações para professor no Telegram.	47
Figura 8 - Interpretações com erros de digitação.....	48
Figura 9 – Ações para aluno no portal acadêmico sagres no aplicativo Messenger.	50
Figura 10 - Requisição de ações do BotUESC no Telegram.	51
Figura 11 - Algumas ações realizadas no site da UESC no Telegram.	52

LISTA DE SIGLAS

AP	Assistente Pessoal
API	Application Programming Interface
BaaS	Backend as a Service
CPU	Central Processing Unit
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IaaS	Infrastructure as a Service
IBM	Internacional Business Machines
IDL	Interface Definition Language
IP	Internet Protocol
PaaS	Platform as a Service
REST	Representational State Transfer
SAP	Software de Assistência Pessoal
SSH	Secure Shell
UESC	Universidade Estadual de Santa Cruz
UML	Unified Modeling Language
URL	Uniform Resource Locator
VPS	Virtual Private Server

SUMÁRIO

1	INTRODUÇÃO	11
1.1	Objetivos	13
1.1.1	Objetivo Geral	13
1.1.2	Objetivos Específicos	14
1.2	Estrutura.....	14
2	EMBASAMENTO TEÓRICO.....	15
2.1	Processamento de Linguagem Natural	15
2.2	Backend as a Service.....	16
2.3	Sistemas Distribuídos.....	17
2.4	VPS Hosting.....	18
2.5	Web Scraping.....	18
2.6	Metodologias Ágeis.....	19
3	MATERIAIS E MÉTODOS	21
3.1	Kanban	21
3.2	Sistema de Extração de Dados	22
3.2.1	Selenium WebDriver.....	22
3.2.2	Python	23
3.2.3	Indexador de Bots	23
3.2.4	BotSagres.....	24
3.2.5	BotUESC	25
3.3	Interface.....	26
3.3.1	Agente de Interface do Telegram.....	28
3.3.2	Agente de Interface do Messenger	29
3.3.3	JavaScript.....	31
3.4	Entendimento de Linguagem Natural.....	31
3.4.1	Watson Assistant.....	32
3.5	Banco de Dados	37
3.5.1	Back4App	37
3.6	Assistente Pessoal.....	40
3.7	Implantação.....	41
3.7.1	Heroku Dyno	41
3.7.2	DigitalOcean Droplet	42
4	RESULTADOS E DISCUSSÕES.....	45
5	CONCLUSÃO	55
5.1	Trabalhos Futuros	55
	REFERÊNCIAS.....	56

1 INTRODUÇÃO

Com a disseminação de computadores pessoais e dispositivos móveis conectados a internet, junto ao avanço da tecnologia cognitiva na área de linguagem natural, *softwares* de assistência pessoal (SAP) surgiram para uma variedade de propósitos, tais como: gerenciamento de trabalho, organização e recuperação de informações e agendamento de atividades (MITCHELL *et al.*, 1994).

O objetivo destes *softwares* é auxiliar ou substituir seus usuários em determinadas tarefas, deixando-os livres para realizar atividades mais importantes. Converging aos conceitos de SAP, expostos por ZAMBIASI e RABELO (2012), Enembreck e Barthes (2002) explicam que o papel principal de um assistente pessoal (AP) é isentar o usuário de realizar tarefas repetitivas ou entediadas. Além disso, os autores deixam claro que as aplicações de um AP podem variar de pesquisas na internet até mesmo tarefas colaborativas.

A interação entre usuário e o AP, normalmente, se dá através da troca de mensagens, onde o usuário faz uma pergunta ou uma requisição de serviço, e o assistente a responde, ou executa o serviço requisitado. Desta maneira, para construção de um SAP é necessário a implementação de um meio de comunicação entre o usuário e o assistente, além da criação de serviços que possam ser requisitados pelo usuário. Numa comunicação realizada por meio da troca de mensagens textuais, fazendo somente o uso da língua natural sem caracteres especiais para identificar comandos específicos, é necessário o entendimento de linguagem natural por parte do assistente, objetivando a compreensão das requisições realizadas pelo usuário, como é mostrado por USACHEV *et al.* (2018) e REATEGUI, RIBEIRO e BOFF (2008), onde os autores desenvolveram em seu módulo de comunicação um sistema para o entendimento de linguagem natural.

Tendo em vista os objetivos de um AP e as suas principais necessidades, em conjunto ao modelo de sistema exposto por REATEGUI, RIBEIRO e BOFF (2008), pode-se realçar que para a implementação de um SAP é indispensável o desenvolvimento de um meio de comunicação, armazenamento de dados, entendimento de linguagem natural e por fim a execução dos serviços necessários. Desta maneira, um SAP pode ser visto como uma grande aplicação monolítica, e a construção de um *software* desta maneira gera acoplamento e dificuldade de manutenção (IBM, [S.d.]). Entretanto, visando evadir-se destes problemas, as

necessidades do SAP podem ser divididas em agentes capazes de executar tarefas específicas, criando assim um sistema multiagente.

Wooldridge (2002, p. XIII) apresenta sistemas multiagente como sistemas compostos por múltiplos elementos computacionais capazes de interagir, conhecidos como agentes. O autor também conceitua um agente como:

Um sistema computacional com dois importantes recursos. Primeiro, eles são pelo menos até certo ponto capazes de ações autônomas – de decidir por si mesmos o que eles precisam fazer para satisfazer seus objetivos. Segundo, eles são capazes de interagir com outros agentes – não simplesmente trocando dados, mas com engajamento em atividades sociais que nós realizamos diariamente em nossa vida: cooperação, coordenação, negociação e coisas do gênero.

Nesse sentido, Reategui, Ribeiro e Boff (2008) propõem um sistema multiagente para o controle de um AP, que é explicado pelos autores: “cada agente controla uma funcionalidade específica, e um agente mediador define qual deles deve entrar em ação a cada momento” (REATEGUI; RIBEIRO; BOFF, 2008).

Desta maneira, pode-se obter soluções adquiridas a partir de ferramentas de terceiros, por exemplo, o recurso de entendimento de linguagem natural pode ser obtido com o IBM Watson, como é apresentado por YAN *et al.* (2016), PITON (2017) e MOSTAÇO *et al.* (2018). Não restrito ao entendimento de linguagem, outros recursos como a interface de comunicação, i.e. o meio de troca de mensagens pode ser modularizada e obtido a partir de outras ferramentas, como é mostrado por MOSTAÇO *et al.* (2018) onde é utilizado a *Application Programming Interface* (API) do aplicativo de mensagens Telegram para a comunicação com o usuário.

A comunicação entre os agentes, ou módulos, de um SAP geralmente é realizada por meio da internet, pois quase sempre os módulos estão sendo executados em computadores e locais distintos. Sendo assim, um SAP implementado de maneira modularizada pode ser considerado um sistema distribuído. Segundo TANENBAUM e VAN STEEN (2013) um sistema distribuído é como uma coleção de computadores independentes que cooperam para resolver uma tarefa, entretanto o usuário final os enxerga como um só. Ainda COULOURIS, DOLLIMORE e KINDBERG (2012) acrescentam ao conceito de sistema distribuído como sendo um sistema no qual *softwares* ou *hardwares* conectados à rede comunicam-se e coordenam as suas ações por meio de troca de mensagens.

Dada a necessidade de disponibilizar serviços pela internet, se faz necessária a criação de serviços de rede (*web services*), ou seja, servidores conectados à internet, construídos com o propósito de suprir as necessidades de um *site* ou uma aplicação. Programas clientes utilizam APIs para se comunicar com estes *web services*. De modo geral, uma API disponibiliza dados e funções para facilitar a interação entre os *softwares* e permite que eles troquem informações. Uma API *web* é a interface de um serviço *web*, que recebe e responde requisições de clientes (MARK, 2013).

Como é exposto por USACHEV *et al.* (2018) um AP pode ser utilizado para obter informações para um usuário. Nesse sentido, um SAP pode utilizar uma API disponibilizada por outro sistema para a obtenção de dados ou para a realização de alguma ação.

Entretanto, nem todos os *sites* existentes fornecem uma API, e para extrair informações destes sistemas *web* uma das principais técnicas é o uso do *web scraping* para obter estes dados de forma automatizada (MALIK; RIZVI, 2011). VARGIU e URRU (2012) apresentam *web scraping* como “[...] uma técnica de *software* destinada a extrair informações de *sites*. *Web scrapers* simulam a exploração humana na internet” (VARGIU; URRU, 2012, tradução nossa). Os autores apresentam o termo *web scrapers*, este é designado a *softwares* programados para buscar, em uma página da *web*, informações e/ou executar ações utilizando a técnica de *web scraping*. Este tipo de *software* automatizado é popularmente conhecido como *um bot scraper*. Tais *bots* percorrem o *site* através do código fonte, normalmente em formato HTML, e também podem executar instruções em JavaScript na página.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é implementar um *software* de assistência pessoal para auxiliar os usuários quanto às informações produzidas no âmbito da Universidade Estadual de Santa Cruz, capaz de receber mensagens textuais, interpretá-las, executar serviços quando necessário, e retornar ao usuário final uma resposta trivial ou um resultado da execução de um serviço requisitado.

1.1.2 Objetivos Específicos

Os objetivos específicos deste projeto são:

- a) determinar as informações que serão disponibilizadas pelo SAP a partir dos serviços de comunicação oferecidos pela UESC;
- b) estabelecer um meio de comunicação entre usuário e assistente pessoal;
- c) estabelecer o entendimento entre as necessidades do usuário e as ações que possam ser realizadas pelo SAP.

1.2 Estrutura

Este trabalho de conclusão de curso foi dividido em 4 tópicos gerais, apresentando-se no primeiro um aprofundamento a respeito da teoria utilizada neste projeto. No segundo tópico é abordado a implementação de cada módulo do projeto, como e quais ferramentas foram utilizadas para o desenvolvimento de cada parte do projeto. O terceiro tópico caracteriza a exposição dos resultados obtidos e uma discussão sobre eles, além de exibir os impedimentos sofridos durante o projeto. Por fim, no ultimo tópico é apresentado as conclusões acerca dos resultados obtidos e proposto implementações futuras.

2 EMBASAMENTO TEÓRICO

Neste tópico será apresentada a fundamentação teórica necessária para o desenvolvimento deste projeto.

2.1 Processamento de Linguagem Natural

O entendimento de linguagem natural, também conhecido como processamento de linguagem natural, é uma área que estuda o desenvolvimento de *softwares* que analisam e reconhecem linguagens humanas, ou linguagens naturais (PERNA; DELGADO; FINATTO, 2010).

Esta área trata computacionalmente os aspectos da comunicação humana considerando estruturas e contextos das sentenças. De uma maneira geral, o objetivo do entendimento de linguagem natural é estabelecer uma comunicação com o computador por meio da linguagem humana (GONZALEZ; LIMA, 2003).

Com o entendimento de linguagem natural os desenvolvedores podem organizar uma estrutura de conhecimento para realizar tarefas como traduções, resumo de textos, reconhecimento de entidades, análise de sentimentos e reconhecimento de fala. Além disso, este processamento de linguagem pode ser utilizado para construção de *bots* que possuam a finalidade de conversar com humanos e até mesmo gerar automaticamente palavras chaves identificando o contexto de um texto (KISER, 2016).

Devido à rica ambiguidade da linguagem natural, o entendimento desta se torna uma tarefa não trivial. Essa ambiguidade faz com que o entendimento de linguagem natural seja diferente do entendimento de linguagens de programação, já que estas foram criadas com definições que evitam justamente a ambiguidade (PERNA; DELGADO; FINATTO, 2010). Para entender a linguagem humana é necessário entender mais que somente as palavras, é preciso entender os conceitos de cada palavra e o contexto criado quando diversas palavras estão juntas em uma mesma sentença (KISER, 2016).

Ainda PERNA, DELGADO e FINATTO (2010) acrescentam que:

Vários métodos ou técnicas computacionais podem ser usados para analisar e interpretar a linguagem natural. Existe uma clara distinção, por exemplo, entre as técnicas empregadas na linguagem falada e na linguagem escrita,

mas também além destas divisões é possível ver diferenças claras entre as técnicas necessárias para diferentes tipos de aplicação, e que podem envolver diferentes tipos de textos, como por exemplo, textos científicos, jornalísticos, literários, etc. Neste sentido, a grande maioria das aplicações é voltada para um tipo específico de linguagem.

Devido a este envolvimento de textos com diferentes aspectos, as empresas que fornecem o serviço de entendimento de linguagem natural, normalmente, requisitam aos clientes que sejam cadastradas quais são as intenções e entidades que devem ser extraídas do texto, como é o caso do Google DialogFlow e IBM Watson Assistant (GOOGLE, [S.d.]) (MILLER, 2017).

As intenções são os objetivos que o usuário terá ao interagir com o serviço, ou seja, as ações que o usuário pretende realizar com o serviço. Para cada intenção é necessário incluir elocuções de amostra que refletem a entrada que os clientes possam usar para obter as informações que eles precisam. Já as entidades representam um termo ou objeto que fornece um contexto para uma intenção, i.e., elas são as entidades que podem aparecer durante a conversação, como locais, datas, horários ou códigos específicos de algum objeto.

2.2 Backend as a Service

Um *Backend as a Service* (BaaS) pode ser visto como um serviço que auxilia a conexão entre o *backend* e o *frontend* de uma aplicação. O BaaS ajuda os desenvolvedores a acelerar o desenvolvimento de *software* e simplificar a criação de APIs. Com ele não é necessário desenvolver todo o *backend* de um aplicativo, apenas utilizar o BaaS para criar as APIs e conectar à aplicação (BATSCHINSKI, 2016).

O BaaS fornece aos desenvolvedores de aplicações *web* e móveis uma maneira de conectar seus aplicativos ao armazenamento e processamento em nuvem e ao mesmo tempo fornecem recursos comuns como gerenciamento de usuários, notificações, integração de redes sociais e outros recursos bastante utilizados por aplicações *web* e móveis. Essa nova geração de soluções BaaS são fornecidas por meio de bibliotecas personalizadas e APIs. Em geral, o BaaS é um serviço relativamente novo na área de computação em nuvem, a maioria das empresas que fornecem essa solução surgiram a partir do ano de 2011 (LANE, 2013).

2.3 Sistemas Distribuídos

Um sistema distribuído é um sistema que é composto por componentes que estão localizados em computadores diferentes, interligados por uma rede, que coordenam as suas ações a partir do repasse de mensagens, entretanto, para o usuário final parecem ser apenas um único sistema. Desta maneira, um sistema distribuído possui concorrência de componentes, falta de um relógio global e falhas de componentes independentes (COULOURIS; DOLLIMORE; KINDBERG, 2012) (TANENBAUM; VAN STEEN, 2013).

Basicamente, um sistema distribuído é composto por diversos componentes, i.e., computadores, autônomos transparentes ao usuário final, de maneira que eles enxerguem todos os componentes como um único. Desta maneira, os componentes precisam cooperar. A maneira na qual esta colaboração é realizada é um dos princípios de sistemas distribuídos (TANENBAUM; VAN STEEN, 2013).

Os computadores envolvidos em um sistema distribuído podem ser dos mais variados tipos e possuir as suas mais variadas diferenças, de *mainframes* a até mesmo pequenos sensores conectados a uma rede. Entretanto, para que haja uma comunicação entre eles é necessário que exista uma *Interface Definition Language* (IDL), um padrão uniforme de comunicação que descreve as interfaces assim como suas assinaturas (TANENBAUM; VAN STEEN, 2013) (MESSERSCHMITT, 1999).

Os sistemas distribuídos podem ser implementados para os mais variados propósitos, por exemplo, a própria internet é um sistema distribuído, assim como jogos online, e-mails, redes sociais entre outros (COULOURIS; DOLLIMORE; KINDBERG, 2012).

Os desafios na criação de um sistema distribuído aparecem devido a necessidade de permitir que novos componentes sejam adicionados ou substituídos, prover segurança em todo o sistema, possuir uma boa escalabilidade, i.e., funcionar bem quando a carga ou o número de usuários aumenta, tratar concorrência entre os componentes e ofertar um serviço de qualidade (COULOURIS; DOLLIMORE; KINDBERG, 2012).

2.4 VPS Hosting

Virtual Private Server (VPS) Hosting é a hospedagem de máquinas virtuais, vendido por empresas como um serviço. Cada VPS possui seu sistema operacional dedicado e os clientes destas empresas possuem acesso de usuário com direitos de administrador destas máquinas. Um VPS pode usar uma *Central Processing Unit* (CPU) compartilhada com outros VPSs, isso significa que cada um receberá uma porção de *threads* compartilhadas de uma CPU virtualizada. Entretanto, os VPSs podem utilizar *threads* dedicadas de uma CPU virtualizada, aumentando assim o seu desempenho e consistência durante processamento intensivo da CPU (DIGITALOCEAN, 2018b).

Um VPS fornece maior custo-benefício em comparação com servidores dedicados, além disso, os VPSs fornecem recursos computacionais previamente garantidos. Até pouco tempo atrás, o *VPS Hosting* não possuía um esquema de arquitetura flexível, a alocação de recursos era realizada no momento da criação do contrato com a empresa e caso o cliente quisesse atualizar algum recurso somente durante um período de pico de uso, ele tinha que pagar até mesmo pelos momentos em que o VPS estava em desuso (TELENYK *et al.*, 2013).

Atualmente, grande parte das empresas que vendem o serviço de *VPS Hosting* fornecem ao cliente a possibilidade de alterar recursos de maneira mais dinâmica por meio de um painel de controle, criando assim a capacidade de alterar os recursos de um VPS somente nos momentos necessários e reduzindo os custos de utilização do serviço (TELENYK *et al.*, 2013).

2.5 Web Scraping

O *web scraping* é uma técnica de *software* que objetiva a extração de informações de páginas da *web*. Normalmente, o uso da técnica consiste na imitação de um ser humano utilizando a internet e interagindo com componentes de sistemas *web* ou até mesmo realizando requisições e enviando informações.

Web scraping está relacionado com uma outra técnica chamada de indexação da *web*, esta é uma técnica adotada por diversos mecanismos de pesquisa para indexar informações da internet por meio de um *bot*. Entretanto, o *web scraping* é mais voltado para transformação de dados não estruturados na internet, geralmente

dispostos em formato HTML, em dados estruturados que possam ser armazenados e analisados (VARGIU; URRU, 2012).

Tal técnica também pode ser utilizada como maneira de automatizar a extração ou inserção de dados em sistemas. Normalmente, é criado um *software* capaz de percorrer estruturas HTML e executar funções JavaScript, estes são conhecidos como *bots scrapers*. Por fim, estes *bots* são utilizados para realizar ações específicas em alguma página para diversas finalidades.

Alguns problemas podem ser enfrentados ao utilizar a técnica de *web scraping*, uma delas é a má formatação HTML de uma página, ao se deparar com esse problema é necessário que o interpretador de HTML usado seja capaz de identificar e contornar este problema para que a execução do *bot* possa continuar sem maiores problemas (PAULO, 2017). Além disso, como o *web scraping* é utilizado para automatizar alguma ação, pode ser encontrado problemas quando não existe padronização para a disposição de elementos HTML na página, quando tal problema é enfrentado o desenvolvedor deve criar ações específicas para cada ação não padronizada, desta maneira, o enfrentamento deste problema acaba gerando mais tempo para a criação de um *bot scraper*.

Atualmente o *web scraping* é comumente utilizado para comparação de preços online, monitoramento de dados meteorológicos, detecção de mudanças em sistemas *web*, buscas na internet, compilação de diferentes fontes de informação em um só lugar e integração de dados da internet (VARGIU; URRU, 2012).

Para o uso da técnica de *web scraping* não é obrigatório o uso de uma linguagem de programação específica, pois somente é necessário a criação de um interpretador de HTML eficiente. Entretanto, as principais linguagens de programação utilizadas para tal técnica são Python, Ruby e Java (PAULO, 2017). Além disso, existem diversas bibliotecas que oferecem maior facilidade na criação de *bots scrapers*, pois estas já fornecem interpretadores e dessa maneira somente é necessário o uso do interpretador para automatizar as ações.

2.6 Metodologias Ágeis

As metodologias ágeis são apontadas como uma alternativa as formas tradicionais de desenvolvimento de *softwares*. Estas são voltadas para projetos em que possam haver muitas mudanças, os requisitos são passíveis de alterações,

modificar partes de códigos não são tarefas de alto custo, as equipes são pequenas, datas de entregas são curtas e é essencial que a implementação seja realizada de maneira rápida sem requisitos estáticos (SOARES, 2004).

Embora todos os métodos ágeis sejam baseados no desenvolvimento e entrega incremental, eles propõem diferentes processos para conseguir isto. Entretanto, eles compartilham o mesmo conjunto de princípios e, portanto, têm muito em comum. Em 2001, dezessete especialistas em processos de desenvolvimento de *softwares* apresentaram um documento que estabelece princípios comuns a serem compartilhados por todos os métodos ágeis. Este documento contendo os princípios é conhecido como manifesto ágil. Os principais conceitos apresentados por ele são: envolvimento do cliente; entrega incremental; membros da equipe devem desenvolver suas próprias maneiras de trabalhar sem processos prescritivos; projetar sistemas que acomode mudanças de requisitos; simplicidade do *software* e do processo de desenvolvimento dele (SOMMERVILLE, 2013) (SOARES, 2004) (BECK *et al.*, 2001).

O manifesto ágil coloca algumas das principais preocupações das metodologias clássicas como tarefas de importância secundária. De acordo com as metodologias ágeis, não é obrigatório a criação de diagramação UML, somente é criado estes diagramas quando é necessário e geralmente de forma automatizada (BECK *et al.*, 2001).

As metodologias ágeis são comumente usadas por pequenas e grandes empresas no cenário atual devido ao seu aumento na velocidade de produção e entrega. Existem diversas metodologias ágeis atualmente, as principais são Scrum, Test Driven Development, Extreme Programming e Kanban (SOMMERVILLE, 2013) (VENTRON, 2018).

3 MATERIAIS E MÉTODOS

Visando a criação de um sistema distribuído, a implementação deste SAP foi modularizada, de forma que cada módulo possuía uma funcionalidade específica. Portanto, foram implementados módulos de interface, banco de dados, entendimento de linguagem natural, assistente pessoal e um sistema de extração de dados.

Antes de dar início ao desenvolvimento do SAP, foi necessário decidir qual tipo de metodologia de desenvolvimento de *software* seria utilizado. Tendo em vista que este projeto poderia sofrer mudanças de requisitos, além da necessidade de diminuir o tempo de entrega das implementações e que para a realização do mesmo não haveria equipe de desenvolvedores, foi decidido que a metodologia de desenvolvimento deveria ser ágil. Em seguida, foi escolhido dentre as metodologias ágeis o Kanban como base de desenvolvimento deste projeto, pois além de ser voltado para pequenas equipes, este auxilia a resolução de problemas trazidos por trabalhos complexos.

3.1 Kanban

O Kanban tem como objetivo aumentar consideravelmente a produtividade dividindo grandes trabalhos em pequenas tarefas, fazendo com que melhore as estimativas de tempo para a finalização de uma tarefa e diminuir os problemas trazidos por um grande trabalho.

Inicialmente o método Kanban foi aplicado em empresas japonesas de fabricação em série. A Toyota, empresa de automóveis, foi a responsável pela introdução do Kanban. Este método permite um maior controle de produção com informações precisas da quantidade de tarefas que foram realizadas, quantas estão em progresso e quantas ainda irão ser iniciadas (7GRAUS, 2015).

Para o funcionamento do Kanban é necessário utilizar um quadro para fixar cartões. Este quadro é dividido em 3 colunas, tarefas a fazer, tarefas em andamento e tarefas terminadas. Cada cartão representa uma tarefa e a cada etapa de produção de uma tarefa esta é movida para a coluna correspondente. Para melhor interatividade com o Kanban foi utilizado a ferramenta de quadro virtual disponível no GitHub, um repositório Git gratuito que oferece diversas ferramentas para auxiliar o desenvolvedor a se organizar.

3.2 Sistema de Extração de Dados

Buscando atender aos objetivos deste projeto, primeiramente, foi necessário definir as informações que seriam ofertadas pelo SAP, portanto, foi preciso realizar o estudo dos sistemas de comunicação da UESC com a comunidade acadêmica. Desta maneira, foi estudado os tipos de informações disponíveis no site da própria universidade e no portal acadêmico sagres, assim como o funcionamento desses sistemas.

Observou-se que ambos são sistemas *web* e, portanto, seria viável extrair dados destes utilizando a técnica de *web scraping*. Desta maneira, foi decidido que seria necessário a criação de dois *bots scrapers* que iriam automatizar a recuperação de informações dos sistemas em questão. Cada *bot* ficaria responsável por extrair dados de um dos sistemas, sendo assim, deu-se início à criação do BotUESC, responsável pela recuperação de informações do site da UESC, e do BotSagres, responsável por extrair informações do portal acadêmico sagres.

Objetivando a redução de acoplamento e dependência entre o módulo de assistente pessoal e os *bots scrapers*, foi criado um módulo indexador de *bots*. A implementação do indexador de *bots* e dos *bots scrapers* foi realizada utilizando a linguagem de programação Python, além disso, visando o melhor aproveitamento da técnica de *web scraping* nos *bots*, foi utilizado o *framework* Selenium WebDriver.

3.2.1 Selenium WebDriver

O Selenium foi escolhido para este projeto devido sua sintaxe simples e a disponibilidade de uma variedade de ferramentas para automação de testes, além de poder ser utilizada em diferentes linguagens de programação.

O Selenium é uma ferramenta criada para testes automatizados de sistemas *web*, porém, o seu uso não se limita a isto, ela também pode ser utilizada para criação de *bots* extratores de informações, visto que é uma ferramenta que pode ser utilizada para a técnica de *web scraping*. Além de possuir uma grande comunidade de desenvolvimento, e uma boa documentação online, o Selenium WebDriver possui bibliotecas para as linguagens de programação: Java, C#, Python, Ruby, PHP, Perl e JavaScript. A biblioteca conta com funções para abrir URLs, clicar em elementos HTML, escrever em caixas de texto, arrastar elementos para algum local da página e

também fornece funções que verificam se um elemento existe ou não, se está visível, ou até mesmo para monitorar o comportamento de algum elemento. Além disso, com o Selenium também é possível executar comandos JavaScript na página *web* (HOLMES; KELLOGG, 2006).

Para utilizar o Selenium WebDriver é necessário, além da instalação do próprio Selenium WebDriver, a instalação de um *Driver* do navegador *web* a ser utilizado, como por exemplo, para utilizar o Google Chrome é necessário o *Chrome Driver* e para o Mozilla Firefox é preciso o *Firefox Driver*. Ao executar o Selenium, é aberto o navegador e realizado automaticamente todas as ações que foram implementadas, estas ações podem ser visualizadas normalmente, pois por padrão a janela do navegador fica aberta, entretanto pode-se configurar o navegador, através do Selenium, para iniciar em modo *headless*, desta maneira o navegador é executado em segundo plano sem interface gráfica.

3.2.2 Python

Python é uma poderosa linguagem de programação bastante flexível e que permite uma rápida prototipação. Ela possui estruturas de dados de alto nível e uma abordagem simples, mas efetiva, de programação orientada a objeto. A linguagem Python tem uma natureza interpretada, ideal para desenvolvimento de aplicações de maneira rápida (SWAROOP, 2003). “Python possui sintaxe simples de fácil de aprendizagem que enfatiza a legibilidade e, portanto, reduz o custo da manutenção do programa.” (PYTHON, 2018, tradução nossa).

O Python foi escolhido para este projeto devido a sua compatibilidade com o Selenium WebDriver, além de permitir a prototipagem de sistemas complexos de forma rápida e flexível e possuir uma variedade de bibliotecas.

3.2.3 Indexador de Bots

O módulo indexador de *bots*, denominado Bothub, é a implementação de um índice de *bots* que realizam ações que possuem o mesmo domínio, i.e., ações que estão relacionadas a um contexto muito próximo. Para este projeto foi implementado somente um indexador, o Bothub UESC, responsável por indexar os *bots* que realizam serviços no contexto da universidade.

Os Bothubs são responsáveis por manter controle das requisições realizadas aos *bots* que estão indexados neles, assim como ter conhecimento de quais são os serviços disponíveis em cada *bot*. Desta maneira, ao receber uma requisição informando qual serviço precisa ser realizado e quais informações devem ser passadas, o Bothub deve ser capaz de encaminhar este pedido ao *bot* que irá atender as necessidades desta requisição.

Ao realizar uma requisição a um Bothub, espera-se sempre que a resposta obtida siga o padrão da IDL estabelecida pela AP, caso o serviço tenha sido executado sem nenhum problema, a resposta obtida deve sempre conter as seguintes informações:

- a) tipo de mensagem (ex.: texto ou imagem);
- b) variável booleana informando se a mensagem possui marcação de estilo;
- c) mensagem.

Caso ao executar um serviço ocorra algum tipo de erro, o Bothub deve responder a requisição com estas informações:

- a) tipo de mensagem (ex.: texto ou imagem);
- b) variável booleana informando se a mensagem possui marcação de estilo;
- c) mensagem;
- d) variável booleana informando que houve um erro ao executar o serviço atribuída com o valor “verdade”;
- e) informações enviadas a requisição que ocasionaram este erro.

Desta maneira, o módulo assistente pessoal não precisa saber qual *bot* é responsável por um serviço específico, mas somente em qual contexto esse serviço se encaixa e requisitar o serviço ao Bothub deste âmbito. Por fim, foi indexado no Bothub UESC: o BotSagres e o BotUESC.

3.2.4 BotSagres

O BotSagres foi implementado visando a realização de ações no portal acadêmico sagres. O desenvolvimento deste *scraper* foi realizado observando o funcionamento do portal e identificando quais ações eram necessárias para extrair os

dados do sistema. Durante o estudo do funcionamento do sagres, foi identificado dois tipos de usuário: professor e aluno. Então foi decidido quais seriam as funcionalidades implementadas para cada um.

As ações disponíveis para os alunos seriam:

- a) calcular o coeficiente de rendimento acadêmico;
- b) enviar uma imagem com os horários de aula;
- c) listar todas as disciplinas cursadas;
- d) listar as disciplinas que estão sendo cursadas;
- e) listar a quantidade de faltas em todas as disciplinas cursadas;
- f) listar a quantidade de faltas em uma disciplina específica;
- g) listar as médias de todas as disciplinas cursadas;
- h) listar os créditos de uma disciplina específica.

Para os professores, foi decidido que as seguintes funcionalidades seriam implementadas:

- a) enviar uma imagem com os horários de aula;
- b) listar as turmas do semestre atual;
- c) informar a quantidade de alunos matriculados em uma disciplina específica;
- d) informar a quantidade de disciplinas que já foram ministradas;
- e) calcular a carga horária semanal ministrada.

Em seguida, foi estudado o que seria preciso para implementar estas funcionalidades. Observou-se que, para realizar qualquer operação no portal é necessário realizar uma autenticação no sistema utilizando o nome de usuário e senha de um aluno ou professor, logo, estas credenciais devem sempre ser informadas ao requisitar a execução de uma das funcionalidades disponíveis. Dentre os serviços oferecidos pelo BotSagres, somente os que precisam de uma disciplina específica necessitam de outras informações, nestes casos é necessário informar o código desta disciplina.

3.2.5 BotUESC

A implementação do BotUESC visa extrair informações diretamente do site da UESC. Similarmente ao BotSagres, o desenvolvimento deste *bot* foi realizado a partir

de observações do funcionamento do sistema na tentativa de descobrir as informações necessárias para realizar cada ação.

Após analisar a relação entre utilidade e viabilidade de desenvolvimento na obtenção de informações no site da UESC, foi decidido a implementação das seguintes funcionalidades:

- a) listar notícias recentes;
- b) listar notícias de um dia específico;
- c) listar resultados recentes;
- d) listar editais recentes;
- e) listar editais de um dia específico;
- f) listar editais de aquisição de bens e serviços recentes;
- g) listar editais de aquisição de bens e serviços de um mês específico;
- h) listar os cursos ofertados pela UESC junto ao *site* do colegiado de cada;
- i) listar os departamentos da UESC e os *sites* de cada.

Observou-se que para realizar algumas ações não seria necessária nenhuma informação adicional, entretanto, serviços que realizam pesquisas em um dia ou mês específico exigiriam que fosse informado pelo usuário a data para realizar tal pesquisa.

A implementação dos *bots scrapers* exigiu a criação de suas respectivas APIs para que o Bothub UESC pudesse realizar requisições diretamente a eles quando necessário. Desta maneira, os *bots* precisaram seguir a IDL previamente estabelecida para que a comunicação entre o indexador e o extrator seja realizada sem problemas, sendo assim, a configuração adotada exige que o ponto de acesso a requisição seja o nome do serviço a ser realizado e as informações passadas pela requisição são os parâmetros necessários para cada ação. Por exemplo, uma requisição ao ponto de acesso “*uesc_listar_ultimos_editais*” deve realizar a ação de listar os últimos editais do site da UESC. Já a resposta enviada ao resolver uma requisição deve seguir o mesmo padrão de resposta do Bothub, apresentado anteriormente no tópico 3.2.3.

3.3 Interface

Em seguida, visando estabelecer uma comunicação entre o AP e o usuário final foi implementado o módulo de interface. Este módulo tem como objetivo lidar com as mensagens de texto recebidas de um usuário. As mensagens recebidas não são

interpretadas neste módulo, apenas são encaminhadas em forma de requisições HTTPS para o módulo assistente pessoal e em seguida a resposta obtida desta requisição é repassada ao usuário. Além da implementação do encaminhamento de mensagem, também foi necessário a criação de um ponto de acesso, este é utilizado pela Lana para enviar mensagens ao usuário assim que algum serviço requisitado finalizar sua execução.

A fim de realizar troca de mensagens entre um agente de interface e o AP, o agente deve seguir o padrão de comunicação estabelecido pela IDL do AP. A requisição para o AP sempre deve conter as seguintes informações:

- a) nome do agente de interface (ex.: Telegram ou Messenger);
- b) nome do usuário;
- c) número de identificação do usuário;
- d) URL do ponto de acesso;
- e) mensagem a ser encaminhada.

E a resposta recebida desta requisição sempre contém as seguintes informações:

- a) tipo de mensagem (ex.: texto ou imagem);
- b) variável booleana informando se a mensagem possui marcação de estilo;
- c) mensagem.

Já o ponto de acesso direto deve receber requisições que contenham as seguintes informações:

- a) número de identificação de usuário destinatário;
- b) tipo de mensagem (ex.: texto ou imagem);
- c) variável booleana informando se a mensagem possui marcação de estilo;
- d) mensagem.

Para o desenvolvimento deste módulo foi proposto o uso de aplicativos de troca de mensagens Telegram e Messenger, visando eliminar a necessidade de criação de uma nova aplicação que serviria apenas como interface, aliado à possibilidade de contatar a AP facilmente a partir de aplicativos com API pública, sem forçar o usuário a adquirir um novo aplicativo para este propósito.

Visando o funcionamento do módulo de interface, primeiramente, foi implementado um agente de interface para o aplicativo de mensagens Telegram. Somente este agente já seria o suficiente para o funcionamento do SAP, entretanto, foi desenvolvido um segundo agente de interface para o aplicativo de troca de mensagens Messenger como prova de conceito de que a implantação de um novo agente de interface não acarretará em nenhuma mudança nos outros módulos já desenvolvidos e para demonstrar a facilidade de migrar ou adicionar novos meios de comunicação ao SAP. Ambos agentes foram desenvolvidos utilizando a linguagem de programação JavaScript.

3.3.1 Agente de Interface do Telegram

O Telegram é um aplicativo popular de troca de mensagens baseado em plataforma de código livre (SUTIKNO *et al.*, 2016). Ele foi escolhido para este projeto pois é uma aplicação gratuita e com uma interface simples, disponível para *smartphones* e computadores pessoais com aplicação *desktop* ou aplicação *web*.

Além disso, o Telegram também disponibiliza uma API para criação de *bots* na plataforma, desta maneira usuários podem interagir com os *bots*, enviando mensagens e comandos. O controle dos *bots* é feito através de requisições HTTPS para a API pública do Telegram.

O Telegram não possui um painel de controle para a configuração de novos *bots*, entretanto, a criação de um novo *bot* no Telegram é feita através de um outro *bot* disponibilizado pelo aplicativo, o BotFather. Ao enviar o comando de criar um novo *bot* para o BotFather, ele pede informações básicas como nome e nome de usuário do *bot* a ser criado e em seguida é informado a chave de acesso, *token*, para controle deste *bot*.

A implementação de novas funcionalidades do *bot* pode ser realizada em qualquer linguagem de programação onde seja possível fazer requisições a API pública do Telegram. Além disso, também pode ser utilizado *frameworks* para a facilitar o uso da API, como por exemplo o TGFancy, *framework* disponibilizado para a linguagem de programação JavaScript.

Este agente de interface foi desenvolvido utilizando o *framework* TGFancy. A escolha de trabalhar a partir de um *framework* e não diretamente com a API do Telegram foi feita pois o TGFancy já implementa alguns tratamentos que são

necessários para o envio de mensagens longas (com mais de 4096 caracteres), imagens ou mensagens com marcação de estilo, diferentemente das requisições diretas a API onde seria preciso implementar esses tratamentos.

O passo seguinte para a implementação foi a criação de um novo *bot*. Foi requisitado ao BotFather um novo *bot* com o nome “Lana” e nome de usuário “lana_pa_bot” e juntamente a confirmação de criação do novo *bot* foi recebido o *token* de acesso dele.

Com o *token* de acesso já informado, foi dado início à implementação das funcionalidades do *bot*, i.e., as funcionalidades necessárias para este *bot* se tornar um agente de interface do SAP. O TGFancy fornece eventos que são ativados ao receber novas mensagens e ao ativar um destes eventos ele pode realizar a chamada de uma outra função passando para ela a mensagem recebida e as informações de usuário.

Foi desenvolvida uma função, chamada “fowardMessageToLana”, que recebe uma mensagem de texto juntamente ao usuário remetente, faz uma requisição a uma API passando a mensagem recebida e por fim envia ao usuário remetente a resposta obtida da API requisitada. A implementação da requisição e recebimento de resposta da API seguiram os padrões estabelecidos pela IDL do AP. Por fim, o evento de recebimento de mensagens, oferecido pelo TGFancy, foi configurado para executar “fowardMessageToLana” ao receber qualquer mensagem.

Após implementar a funcionalidade de encaminhamento de mensagens, foi criado um ponto de acesso chamado “sendMessageEndpoint”, este recebe uma requisição, que segue o padrão da IDL do AP, encaminha a mensagem recebida na requisição para o usuário de destino e por fim responde se houve algum erro ao encaminhar a mensagem.

3.3.2 Agente de Interface do Messenger

Messenger é o aplicativo da rede social Facebook para a troca de mensagens entre os seus usuários. Ele está disponível para acesso via aplicação *web* ou aplicativo móvel. O Messenger foi escolhido para este projeto devido a sua popularidade gerada pelo Facebook, além disso, ele também é totalmente gratuito e possui API pública.

Para a criação de um novo *bot* na plataforma do Messenger, primeiro é necessário criar uma página no Facebook, em seguida, essa página é associada a

uma nova aplicação através do painel de controle disponibilizado pelo Facebook. Após a criação da aplicação pelo painel é necessário a configuração de chaves de autenticação para que o *bot* tenha acesso as funcionalidades do Messenger.

Com o Messenger devidamente configurado através do painel de controle do Facebook, finalmente pode-se dar inicio ao desenvolvimento das funcionalidades do *bot*. Para isto é necessário configurar um *webhook*, um ponto de entrada na *web* que recebe requisições HTTP/HTTPS de outros sistemas ou clientes. Este *webhook*, que obrigatoriamente deve estabelecer comunicações utilizando o protocolo HTTPS, será utilizado pelo Messenger para repassar as mensagens recebidas ao *bot*, assim como enviar mensagens ao cliente. Assim como no Telegram, a criação do *webhook* e suas funcionalidades podem ser realizadas em qualquer linguagem que possua funcionalidade de realizar e receber requisições pela *web* e também é possível utilizar *frameworks* para facilitar o uso da API do Messenger, como por exemplo o BootBot que é disponibilizado para a linguagem de programação JavaScript.

Para criar o agente de interface do Messenger, o primeiro passo realizado foi a criação de uma página no Facebook, o nome dado a página foi Lana. A partir do painel de controle do Facebook foi configurado um *webhook* para receber mensagens dos usuários para a página a partir do Messenger.

Ao finalizar a configuração do *webhook* no painel do Messenger, foi iniciada a implementação das funcionalidades requisitadas para que o *bot* se torne um agente de interface. Similar ao TGFancy, o BootBot também fornece eventos de recebimentos de mensagem com algumas diferenças. Ao receber uma mensagem pelo evento, esta não possui informações sobre o usuário remetente, entretanto, o evento recebe um outro objeto além da mensagem em si, o “chat”. O objeto “chat” possui diversos métodos relacionados a atual conversa, entre eles o método “getUserProfile” que retorna o perfil do usuário que enviou aquela mensagem, além disso o “chat” fornece métodos de enviar novas mensagens ao usuário naquela conversa.

Ao receber uma nova mensagem, este evento é ativado e então o *bot* usa o método “getUserProfile” do objeto “chat” para buscar mais informações sobre o remetente da mensagem. Em seguida é chamada uma nova função, de nome “forwardMessageToLana”, que tem o mesmo objetivo da função “forwardMessageToLana” implementada no agente do Telegram. Entretanto, a função implementada no agente do Messenger recebe não somente a mensagem e o usuário,

como também recebe o objeto “chat” para continuar a conversação sem a necessidade da criação de um novo objeto.

Similar à implementação do agente do Telegram, neste agente de interface também foi criada uma função “sendMessageEndpoint”, que tem o mesmo propósito: criar um ponto de acesso para o envio de mensagens da AP ao finalizar algum serviço para o usuário.

A implementação deste agente de interface também seguiu os padrões estabelecidos pela IDL da AP, a fim de manter um meio de comunicação funcional entre os módulos.

3.3.3 JavaScript

O JavaScript é uma linguagem de programação de alto nível, dinâmica, não tipificada, interpretada, que fornece todos os recursos necessários para ser orientada a objeto e funcional (FLANAGAN, 2011). Originalmente ela foi criada como parte dos navegadores *web* para execução de instruções *client-side*, utilizando o interpretador de JavaScript do Google conhecido como V8.

Entretanto com o objetivo de tornar JavaScript uma linguagem que possa ser executada *server-side* foi criado o Node.JS, um interpretador *server-side* de JavaScript, baseado na implementação do V8, com ênfase em fornecer um bom desempenho e baixo consumo de memória (TILKOV; VINOSKI, 2010).

Com o Node.JS é possível criar aplicações *web* ou desktop inteiramente em JavaScript, sem a necessidade de uma outra linguagem *server-side*.

A linguagem JavaScript foi escolhido para este projeto devido a possibilidade de uso dos *frameworks* TGFancy e BootBot disponíveis para a linguagem, o seu alto desempenho e a abstração de instruções mais complexas, visto que esta é uma linguagem de alto nível.

3.4 Entendimento de Linguagem Natural

Como a comunicação entre usuário e assistente pessoal será realizada por meio da troca de mensagens de texto com linguagem humana, faz-se necessário estabelecer o entendimento entre as necessidades do usuário e as ações que possam ser realizadas pelo SAP por meio do processamento de linguagem natural.

O módulo de entendimento de linguagem natural tem como objetivo interpretar mensagens textuais, recebidas pelos usuários do SAP, identificando as intenções do usuário com o serviço de assistência pessoal e manter um fluxo contextual de conversação. A implementação deste módulo foi realizada a partir do serviço Watson Assistant.

3.4.1 Watson Assistant

Anteriormente conhecido como Watson Conversation, em 2016, foi criado o serviço Watson Assistant. Disponibilizado pelo IBM Watson por meio da plataforma IBM Cloud, com este serviço é possível construir soluções que entendam linguagem natural e responda de maneira similar a uma conversa entre humanos (MILLER, 2017).

Serviços similares são disponibilizados por outras empresas, como, por exemplo, o Google DialogFlow, Amazon Comprehend, TextRazor e outros, entretanto, para este projeto optou-se pelo IBM Watson Assistant, pois este apresentou uma interface mais interativa e bem documentada para a configuração do serviço, em conjunto a possibilidade de criar uma conta gratuita para utilizar o sistema.

O funcionamento proposto pelo serviço é que a interação com o usuário seja realizada através da interface de algum meio de comunicação implementada pelo desenvolvedor, como por exemplo uma janela de bate-papo simples. O Aplicativo envia a mensagem de texto do usuário para um *workspace* do serviço, o Watson Assistant interpreta a entrada do usuário, direciona o fluxo da conversa, reúne as informações necessárias e retorna estes dados à aplicação. Por fim, o aplicativo pode interagir com seus outros sistemas com base no entendimento da intenção do usuário e utilizar as informações adicionais obtidas para, por exemplo, abrir chamados, atualizar informações de conta ou realizar pedidos.

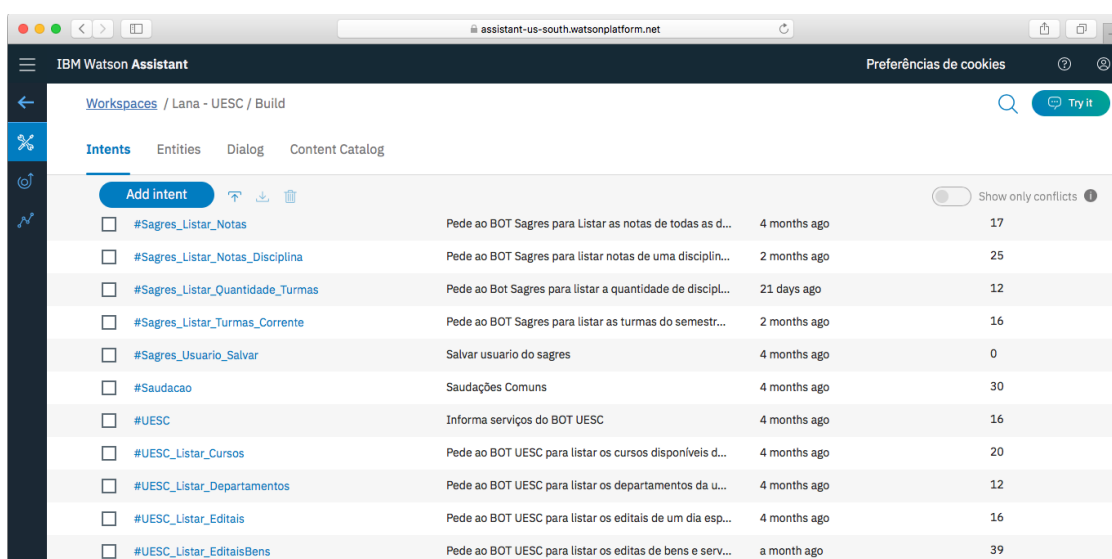
Para a utilização do serviço é necessário a criação de um *workspace*, assim como suas *Intents* e *Entities*. A configuração de *workspace* é realizada através de um painel disponível na plataforma IBM Cloud, neste é possível criar, editar ou remover *workspaces*.

Após a criação de um novo *workspace* é necessário configurar os dados de treinamentos, *Intents* e *Entities*, e o fluxo de diálogo das conversações, *Dialogs*.

As *Intents* são os objetivos que o usuário terá ao interagir com o serviço, i.e., as ações que o usuário pretende realizar com o serviço. Para cada *Intent* é necessário incluir elocuições de amostra que refletem a entrada que os clientes possam usar para obter as informações que eles precisam.

Inicialmente foram cadastradas as *Intents* que demonstravam ações triviais em conversas, como agradecimentos, negações, afirmações e saudações. Em seguida foram cadastradas as *Intents* que representavam os serviços disponíveis pela Lana, estas têm como objetivo identificar nas mensagens a intenção do usuário requisitar ao SAP a execução de algum serviço. Além das *Intents* de serviço, também foram criadas *Intents* para o registro do usuário ao SAP. Algumas podem ser vistas na Figura 1.

Figura 1 - Painel do serviço Watson Assistant com algumas Intents criadas para o workspace Lana.



Intent	Description	Created	Count
#Sagres_Listar_Notas	Pede ao BOT Sagres para Listar as notas de todas as d...	4 months ago	17
#Sagres_Listar_Notas_Disciplina	Pede ao BOT Sagres para listar notas de uma disciplin...	2 months ago	25
#Sagres_Listar_Quantidade_Turmas	Pede ao Bot Sagres para listar a quantidade de discipl...	21 days ago	12
#Sagres_Listar_Turmas_Corrente	Pede ao BOT Sagres para listar as turmas do semestr...	2 months ago	16
#Sagres_Usuario_Salvar	Salvar usuario do sagres	4 months ago	0
#Saudacao	Saudações Comuns	4 months ago	30
#UESC	Informa serviços do BOT UESC	4 months ago	16
#UESC_Listar_Cursos	Pede ao BOT UESC para listar os cursos disponíveis d...	4 months ago	20
#UESC_Listar_Departamentos	Pede ao BOT UESC para listar os departamentos da u...	4 months ago	12
#UESC_Listar_Editais	Pede ao BOT UESC para listar os editais de um dia esp...	4 months ago	16
#UESC_Listar_EditaisBens	Pede ao BOT UESC para listar os editais de bens e serv...	a month ago	39

Fonte: elaborado pelo autor.

Já as *Entities* representam um termo ou objeto que fornece um contexto para uma *Intent*, i.e., elas são as entidades que podem aparecer durante a conversação, como locais, datas, horários ou códigos específicos de algum objeto. O serviço já disponibiliza algumas *Entities* de sistema, para identificar números, datas, percentagens, dinheiro e tempo. As *Entities* possuem valores associados à sinônimos ou padrões, sendo assim, ao cadastrar uma nova *Entitie* é preciso informar os sinônimos, i.e., as maneiras como um valor pode aparecer durante o dialogo, ou uma expressão regular que padroniza este valor.

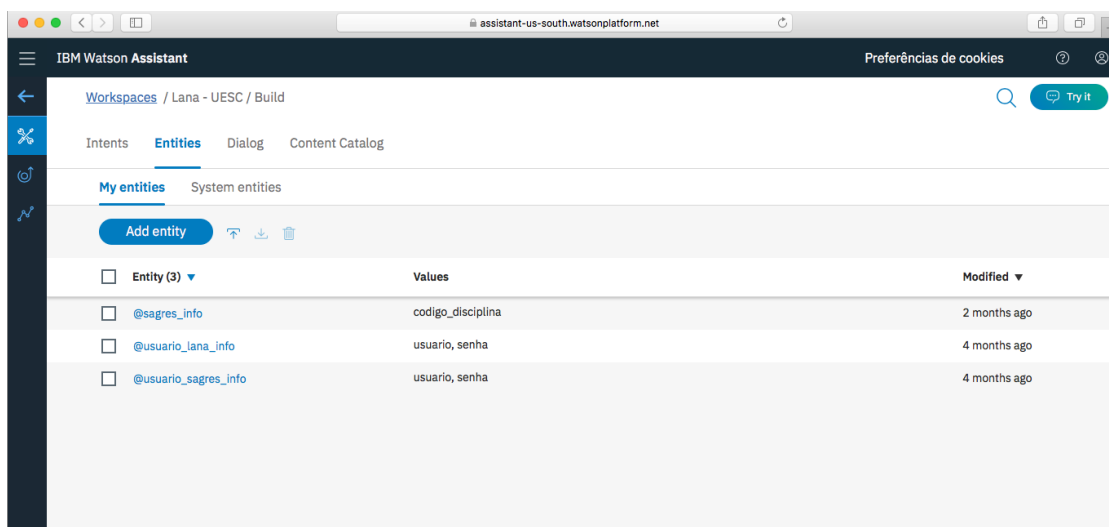
Por exemplo, uma *Entitie* “aeroportos_brasil” pode ser criada para identificar siglas dos aeroportos brasileiros em uma conversação, nesta *Entitie* seria cadastrado o valor “aeroporto de Ilhéus” associado ao sinônimo “IOS”, desta maneira ao receber a mensagem “Chegarei no dia 20 de dezembro em IOS” o Watson Assistant reconhecerá na mensagem a data 20 de dezembro do ano corrente, a partir da *Entitie* de sistema, e o aeroporto de Ilhéus a partir da *Entitie* cadastrada. Um outro exemplo é a criação de uma *Entitie* para a identificação de informações pessoais, neste caso seria incluído o valor como “telefone” e ao invés do uso de sinônimos é utilizado o reconhecimento de padrões, sendo assim é criado uma expressão regular que reconheça um número de telefone e ao enviar uma mensagem contendo um número de telefone qualquer para o serviço este valor é reconhecido podendo ser armazenado no contexto.

Foram cadastradas novas *Entities* para identificar informações necessárias à execução de um serviço e para o cadastro de usuário do SAP.

Foi enfrentado um problema para identificar disciplinas da universidade pelo nome quando inserida em uma mensagem de texto sem nenhum tipo de marcação, visto que a UESC possui diversas disciplinas e cada uma tem um nome diferente, assim como os usuários poderiam escrever o nome de uma mesma disciplina de forma diferente ou abreviada. Uma solução possível seria a criação de uma *Entitie* para cada disciplina e nessa seria cadastrada as formas diferentes de escrever o nome da matéria, entretanto esta solução fica inviável devido ao grande número de disciplinas por curso da UESC e pelo número limite de *Entities* por *workspace* no plano gratuito do Watson Assistant, desta maneira, foi necessário utilizar o código da disciplina para a identificação de uma menção a uma matéria em uma mensagem de texto, já que os códigos de disciplinas da universidade seguem um padrão de no mínimo 3 e no máximo 4 letras seguidas de 3 números.

Portanto, foram criadas as *Entities* para reconhecer códigos de disciplinas da UESC “sagres_info”, assim como, nome de usuário e senha do portal Sagres “usuario_sagres_info” e do próprio SAP “usuario_lana_info”, estas podem ser observadas na Figura 2.

Figura 2 - Painel do serviço Watson Assistant com todas as Entities criadas para o workspace Lana.



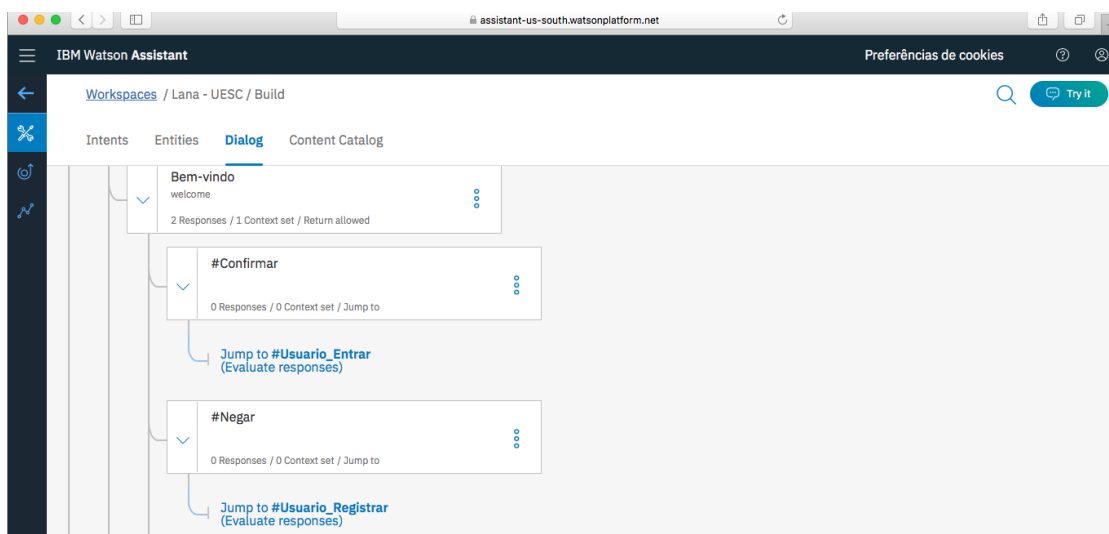
Fonte: elaborado pelo autor.

Conforme são incluídos novos dados de treinamento, um classificador de língua natural é automaticamente incluído no *workspace* e é treinado para entender as solicitações indicadas.

Para finalizar a configuração do *workspace* é necessário montar fluxos de diálogos, ou *Dialogs*, que incorporam as *Intents* e *Entities*. O fluxo de diálogo é representado graficamente na ferramenta como uma árvore, sendo possível incluir novas ramificações para processar cada uma das *Intents* incluídas. Também é possível incluir nós de ramificações que tratam diversas permutações possíveis de um pedido com base em outros fatores, como por exemplo, entidades localizadas na mensagem de entrada ou informações extras passadas ao requisitar o serviço.

Os *Dialogs* foram implementados objetivando a criação do fluxo de conversação de acordo com cada *Intent* identificada nas mensagens textuais. Primeiramente foram implementados os fluxos de conversa para o cadastro ou *login* de um usuário, Figura 3, o nó responsável pelo início deste fluxo de conversa é o “Bem-vindo”, este é o nó criado por padrão que sempre é primeiro a ser executado em novas conversações.

Figura 3 - Painel do serviço Watson Assistant com parte do fluxo de primeira conversa criada para o workspace Lana.



Fonte: elaborado pelo autor.

Ao entrar no nó “Bem-vindo” é perguntado ao usuário se ele já possui uma conta com o SAP, caso seja confirmado a existência de uma conta então o fluxo muda para o nó responsável pela conversação com usuários que desejam fazer *login*, nó “#Usuario_Entrar”, caso seja negada a existência de uma conta então o a conversa será movida para o nó responsável por criações de novas contas, nó “#Usuario_Registrar”.

As criações dos nós responsáveis pela identificação de intenções triviais foram realizadas em seguida. Todos estes nós possuem ações parecidas, onde somente é necessário identificar a intenção do usuário e responder algo que tenha sentido à sua intenção de pergunta. Por exemplo, a criação do nó de saudação somente necessita responder uma outra saudação ao usuário, assim como um nó que identifica a intenção de despedida precisa somente responder mensagem que contenha uma despedida.

Entretanto, foi enfrentado um problema ao identificar a intenção de um usuário realizar algum serviço do BotSagres, pois, era sempre necessário buscar as informações de autenticação do portal Sagres do cliente e só depois realizar o serviço, apesar do Watson Assistant dar suporte a mudança de contextos e pulos entre nós, só foi possível resolver este problema manualmente no módulo de assistente pessoal.

Ao finalizar a criação dos fluxos básicos de conversa, foram implementados os nós responsáveis por gerar respostas quando identificada a necessidade de execução de um serviço, e para cada serviço disponível foi criado um novo nó.

As implementações de respostas dos serviços foram realizadas de maneira diferente das respostas dadas a *Intents* triviais, todos os serviços implementados respondem textualmente ao usuário. Entretanto, junto com a mensagem são enviadas algumas informações necessárias para a execução do serviço, sendo estas:

- a) nome do serviço;
- b) *entities* necessárias para a realização do serviço;
- c) nome do Indexador de Bot responsável pela execução do serviço.

Após realizar toda a configuração do *workspace*, é possível utilizar o serviço Watson Assistant. Ao requisitar o serviço enviando uma mensagem de texto, o Watson vai avaliar a entrada buscando contextos utilizando os *Dialogs* e retornar quais são as *Intents* e *Entities* identificadas na mensagem, a sua taxa de confiabilidade para cada e um objeto chamado “context” que serve para auxiliar o Watson Assistant a manter o contexto das conversações.

3.5 Banco de Dados

Visto que o SAP precisaria guardar alguns dados pessoais de seus usuários, como credenciais do portal acadêmico sagres, foi criado o módulo de banco que tem como objetivo armazenar informações referentes aos usuários do SAP e armazenar os objetos “context” das conversações, visando auxiliar o módulo de entendimento de linguagem natural a manter o fluxo das conversações de diferentes usuários.

Para a implementação deste módulo foi utilizado a plataforma Back4App, pois esta proporciona facilidade e conforto para criar e gerenciar bancos de dados em nuvem.

3.5.1 Back4App

Back4App é uma plataforma BaaS, baseada no Parse, um *framework* popular de *backend*, onde é possível criar e hospedar APIs para aplicações *web* e móveis de maneira mais rápida. O Back4App cria toda a estruturação de banco de dados

MongoDB e sua API, além de outros recursos mais avançados para facilitar o gerenciamento de sistemas e acelerar o desenvolvimento. O sistema é bem documentado e fornece uma enorme facilidade, comodidade e também é gratuito, portanto, estes foram os principais motivos pela escolha de uso do Back4App neste projeto.

A plataforma possibilita a criação de *Cloud Functions*, funções criadas pelo desenvolvedor que são armazenadas no Back4App e ficam disponíveis para serem executadas dentro da própria plataforma, sem a necessidade de criar uma API para realizar tais tarefas ou executá-las localmente. Além disso o Back4App conta com um painel de controle, para gerenciamento de banco de dados, gerenciamento de *Cloud Functions*, controle de *logs*, configurações do servidor, envio de notificações entre outros diversos serviços ofertados.

Utilizando o painel de controle é possível criar novas classes para o banco de dados, novos campos para as classes e adicionar regras de segurança. O Back4App cria automaticamente uma classe de usuário já pré-configurada, e esta possui um campo criptografado para senha, um campo de email e um campo de nome de usuário. Além disso, todas as classes criadas possuem um campo definido automaticamente que armazena data de criação do objeto e um campo que armazena a data da última modificação, os valores atribuídos a estes campos são definidos automaticamente.

A integração com a plataforma pode ser realizada a partir das bibliotecas do Parse disponibilizadas para diversas linguagens de programação como JavaScript, PHP, C# e C, ou por meio de requisições HTTP/HTTPS através da REST API oferecida pelo serviço.

O uso de uma biblioteca cria maior facilidade ao lidar com o uso da plataforma, estas já possuem funções prontas para criação de novos objetos de uma classe, assim como alterar os campos deste objeto, criar novos campos e por fim enviar tudo ao banco de dados para ser armazenado. Além disso, a biblioteca conta com funções específicas para tratar com a classe de usuário, como funções de registro de novos usuários, autenticação e recuperação de senha. Por fim, a biblioteca oferece maior facilidade para a criação de rotinas de banco de dados e adição de novas funções para a API do banco. Estas funcionalidades não são exclusivas às bibliotecas, elas também podem ser utilizadas a partir da REST API, entretanto, será necessário a criação de novas funções para realizar estas requisições.

O banco de dados provido pela pelo Back4app é o MongoDB, um banco de dados não relacional. O MongoDB apresenta em sua documentação dois tipos de modelagem de dados, o modelo de dados incorporado onde as classes criadas possuem todos o conteúdo sem necessitar acessar outras classes, e o modelo de dados normalizado, onde existe a separação de conteúdo entre as classes e é criado um esquema de relacionamento, parecido com a relação chave primária e estrangeira, apresentado pelos bancos de dados relacionais. Para este projeto foi utilizado o modelo de dados incorporado, onde as classes devem possuir todos os atributos necessários para uma consulta sem a necessidade de acessar outras classes através de referências.

Para dar início à implementação deste módulo, primeiramente, a partir do painel de controle do Back4App foi criado um novo projeto “Iana”. Para ajustar o projeto as necessidades do AP, a classe de usuário, criada por padrão, foi modificada adicionando um novo campo “nome” com o objetivo de armazenar os nomes dos usuários do SAP. Por fim, ainda no painel de controle, foi criada uma nova classe, chamada “Context”; nesta classe somente foi necessário criar um campo de nome “context” que armazena um valor do tipo objeto, com o objetivo de atribuir a este campo as estruturas de contexto das conversações.

Após ajustar as configurações do Back4App pelo painel de controle, deu-se início à implementação das *Cloud Functions*, utilizando a linguagem de programação JavaScript, necessárias para o controle das operações no banco de dados. Foram criadas funções com o objetivo de obter um usuário a partir de um valor de um campo, adicionar, modificar ou remover o valor de um campo de um usuário, registrar novos usuários, fazer o *login* de um usuário, criar ou alterar um “context”, obter um “context” a partir de um número de identificação única de interface de um usuário e por fim foi criada uma rotina que é executada a cada 24 horas que tem como objetivo apagar do banco de dados informações que contenham senhas do usuário de sistemas de terceiros, como por exemplo uma senha de usuário do portal Sagres.

Apesar de não ter previamente configurado nenhuma conexão entre a classe “Context” e a classe de usuário, uma relação entre elas é criada a partir do momento que é registrado um novo usuário no banco de dados. Ao registrar um novo usuário, utilizando a *Cloud Function* responsável, é criado um novo campo na tabela de usuário. Este campo é construído para ter o nome do agente de interface que o usuário está utilizando e o valor atribuído a este campo é o número identificador único

informado pelo agente de interface. Por exemplo, se um usuário entrar em contato a partir do Telegram é criado um novo campo na classe usuário chamado “telegram” e o valor deste campo é atribuído para esse usuário como o número de identificação informado pelo agente de interface do Telegram, mesmo com a comunicação dos dois módulos não sendo realizadas diretamente, a Lana repassa essas informações a este módulo.

Similarmente a criação de um novo usuário, ao criar uma nova entrada para a classe “Context” a *Cloud Function* implementada também adiciona um novo campo a esta classe com o nome do agente de interface responsável por aquela conversação e o valor atribuído é o número de identificação única do usuário que possui o objeto “context” a ser armazenado.

Desta maneira, ao procurar por um contexto de conversa ou procurar informações de um usuário, é possível buscar a partir do número de identificação único do agente de interface e obter as informações referentes ao mesmo usuário em classes diferentes.

Por fim, todas as *Cloud Functions* criadas foram implantadas no Back4App e podem ser executadas via requisições HTTPS utilizando a REST API fornecida pelo próprio Back4App, a partir do painel de controle da plataforma ou utilizando o método “run” do objeto “Cloud” da biblioteca de desenvolvimento do Back4App.

3.6 Assistente Pessoal

A fim de estabelecer uma lógica de operação entre todos os módulos, foi criado o módulo de assistente pessoal, isto é, a implementação da assistente pessoal, nomeada como Lana. Este módulo é responsável por estabelecer comunicação com os outros módulos e realizar toda a lógica de operação para o funcionamento da AP. Além disso, suas ações são baseadas nas respostas obtidas pelo módulo de entendimento de linguagem natural. Desta maneira, ele é responsável pela decisão de criar novos usuários, inserir novas informações no banco de dados e iniciar serviços. A implementação deste módulo foi realizada com a linguagem de programação JavaScript.

Primeiramente, foi necessário desenvolver métodos para criar um meio de comunicação entre todos os módulos do sistema. Foram criadas classes que serviram como fachada para acessar funções específicas dos módulos de banco de dados e

de entendimento de linguagem natural. O funcionamento do assistente pessoal segue um fluxo lógico de comunicação, ao receber uma requisição de um módulo de interface o assistente envia a mensagem recebida ao módulo de entendimento de linguagem natural e baseando-se na sua resposta decide se é necessário armazenar uma nova informação no banco de dados, dar início a um serviço, criar uma nova conta ou não realizar nenhuma ação e somente responder de volta ao usuário.

Caso a mensagem recebida tenha a intenção de iniciar um novo serviço, a Lana responde ao usuário informando que o serviço requisitado será realizado e faz uma requisição ao módulo indexador de *bots*, o Bothub, pedindo a execução de um serviço, passando a ele as informações necessárias para o funcionamento deste. Por fim, ao receber a resposta do Bothub, a Lana encaminha a resposta do serviço realizado, através do ponto de acesso direto do agente de interface, ao responsável pela comunicação com o usuário requisitante.

As requisições realizadas a um Bothub devem seguir ao padrão estabelecido pela IDL. A requisição deve conter sempre o nome do serviço a ser executado e caso haja alguma informação adicional esta deve ser passada juntamente a requisição.

Ao finalizar a implementação do módulo de assistente pessoal, foi criada uma API para que os agentes do módulo de interface pudessem realizar requisições a Lana.

3.7 Implantação

Com todos os módulos devidamente implementados, fez-se necessário colocá-los na internet para que eles estejam sempre disponíveis para requisições e assim, possam estabelecer uma comunicação. Desta maneira, os módulos foram implantados em Heroku Dynos e DigitalOcean Droplet.

3.7.1 Heroku Dyno

“Heroku é uma plataforma de nuvem baseada em um sistema de contêiner gerenciado, com serviços e dados integrados e um poderoso ecossistema, para implementar e executar aplicativos modernos.”(HEROKU, 2018). O Heroku oferece um tipo de VPS, o Dyno. Ele é mais limitado que um DigitalOcean Droplet, entretanto

com o Dyno não é necessário desenvolver toda a estruturação de um servidor *web*, somente a integração da aplicação nele.

Um Dyno oferece facilidade ao desenvolvedor para lançar e manter serviços *web*, ele é um VPS já configurado como um servidor *web*, no qual somente é necessário executar nele uma aplicação *web* que receba requisições em uma porta específica, esta porta é alterada sempre que ocorre alguma modificação na aplicação que esta sendo executada, ou sempre que o Dyno é reiniciado, o número da porta fica disponível no próprio servidor armazenado em uma variável de ambiente de nome “PORT”.

Todas as configurações do VPS podem ser acessadas através do painel de controle oferecido pelo Heroku. A partir deste painel é possível adicionar configurações de linguagens de programação que serão utilizadas no Dyno, adicionar novas variáveis de ambiente, monitorar *logs* e atividade do servidor, entre outras opções.

A criação da aplicação é realizada localmente, e ao finalizar a implementação é enviado então o código fonte ao Dyno, utilizando o Git, e a aplicação enviada será então executada no VPS. O Heroku deixa o Dyno em repouso enquanto não existem novas requisições destinadas a ele, quando uma nova requisição chega então o Dyno é iniciado novamente para atender ao pedido. Ao enviar uma aplicação para o VPS, é informada uma URL ao usuário, por fim, as requisições HTTPS para o serviço podem ser realizadas para o Dyno por meio desta URL.

A facilidade para subir novos serviços para a *web*, sem a necessidade de configuração de um servidor para aplicações mais simples, em conjunto a gratuidade do Heroku Dyno foram os motivos para a escolha desta ferramenta.

Os agentes do módulo de interface, assim como, a Lana e o BotHub UESC foram implantados em um Heroku Dyno, visto que estes não necessitam de uma configuração mais detalhada de um ambiente de produção.

3.7.2 DigitalOcean Droplet

A empresa DigitalOcean fornece um serviço de VPS *Hosting* chamado DigitalOcean Droplet. Um Droplet é um VPS com recursos adicionais de armazenamento, segurança e monitoramento para executar facilmente os aplicativos em produção (DIGITALOCEAN, 2018a).

A DigitalOcean fornece o VPS com o sistema operacional já instalado, entretanto, para utilizar o Droplet é necessário configurá-lo de acordo com a sua finalidade, i.e., para utilizar o VPS como um servidor *web*, por exemplo, é responsabilidade do cliente instalar as ferramentas necessárias, assim como configurar o Droplet para receber requisições e atualizações de *softwares*.

Ao criar um Droplet, o DigitalOcean atribui a ele um novo endereço IP e um usuário e senha de administrador, estes podem ser utilizados para estabelecer uma conexão SSH com o VPS, tendo assim acesso a máquina e podendo configurá-la como desejar. Além disso, o DigitalOcean fornece um painel de controle, onde é possível monitorar o uso de atributos do Droplet, assim como realizar algumas configurações como alteração do tamanho da memória, de CPUs, desligar o VPS ou até mesmo mudar o sistema operacional utilizado.

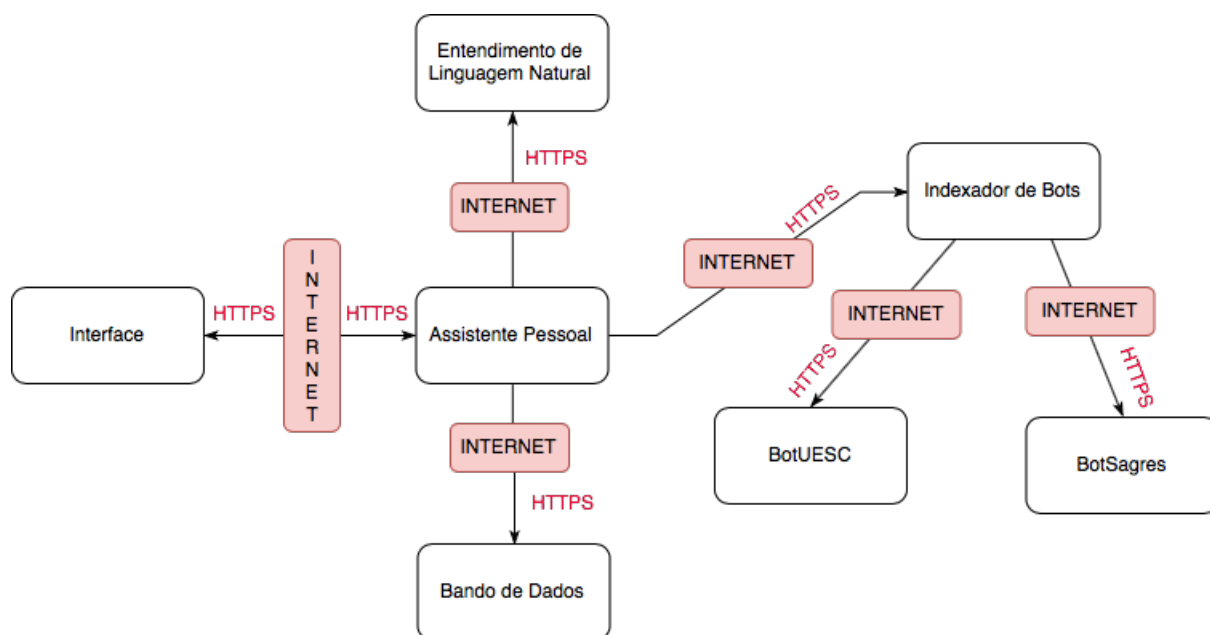
Apesar das tarefas de configuração e manutenção que um Droplet precisa, este continua sendo uma boa proposta pois oferece maior liberdade ao usuário quando se necessita instalar ferramentas que possuem restrições de sistema operacional ou dependências. Esta liberdade, aliada ao fato de ter conseguido um Droplet gratuito foram os motivos da escolha do DigitalOcean neste projeto.

Para o funcionamento do Selenium, como já explicado no tópico 3.2.1, é necessário instalar um *driver* do navegador *web* a ser utilizado. Este processo realiza algumas ações que necessitam de maior controle do ambiente que está sendo configurado, por este motivo o BotUESC e BotSagres foram implantados em um DigitalOcean Droplet.

Ao finalizar a implantação de todos os módulos, a Lana já estava em funcionamento com todos os seus serviços disponíveis para acesso ao público a partir do *bot* do aplicativo de mensagens Telegram. A topologia de comunicação entre os módulos é representada na Figura 4, nela podemos observar que todas as comunicações entre os módulos são realizadas através da internet utilizando o protocolo HTTPS.

A Figura 4 deixa claro que o módulo de interface realiza requisições ao módulo de assistente pessoal, assim como o módulo de assistente pessoal realiza requisições ao de interface, entretanto, somente o assistente realiza requisições aos outros módulos, exceto aos *bots scrapers* onde as requisições a eles são realizadas pelo indexador, e estes outros módulos não realizam requisições ao assistente, somente enviam respostas.

Figura 4 - Topologia de comunicação entre módulos.

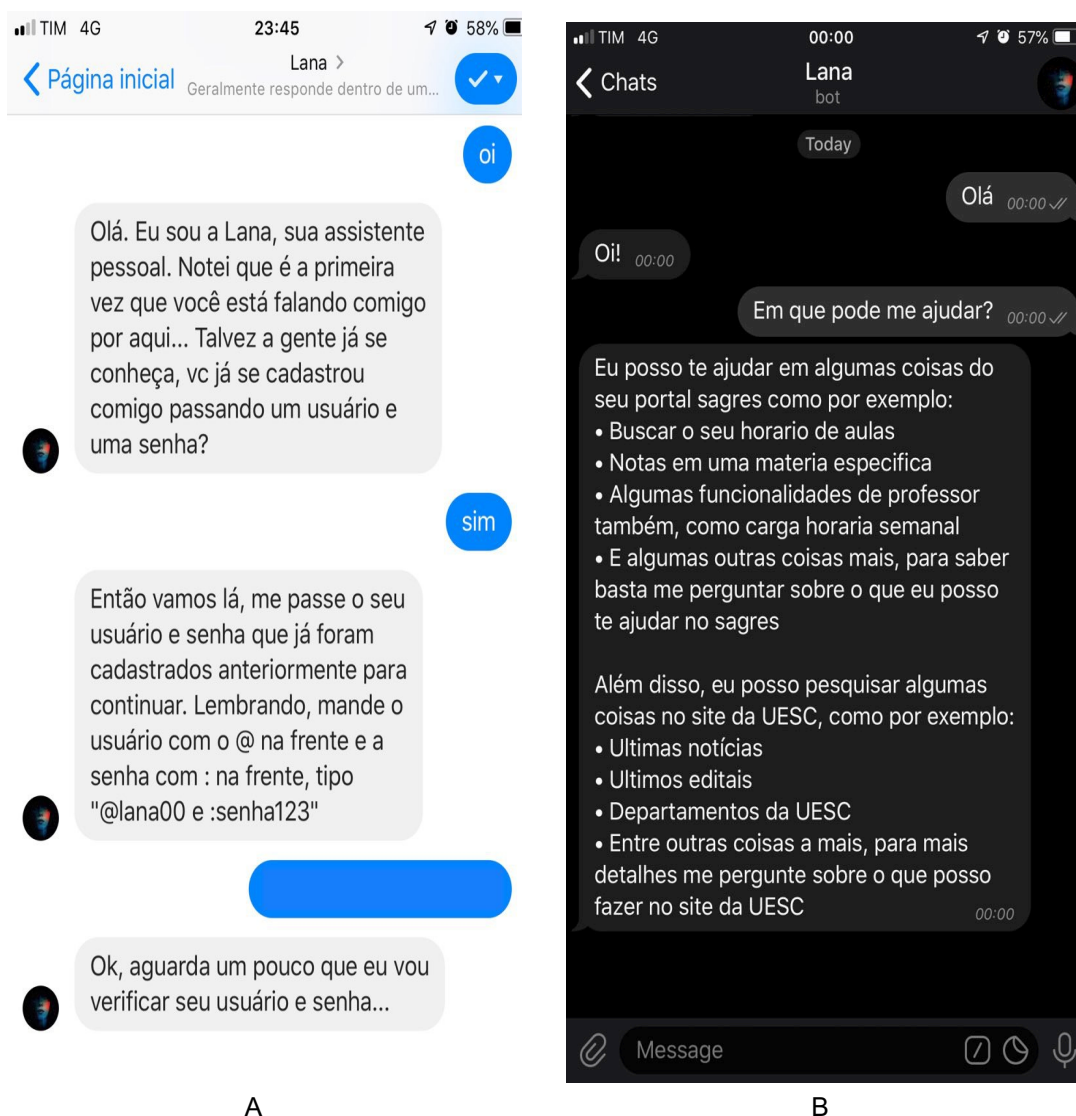


Fonte: elaborado pelo autor.

4 RESULTADOS E DISCUSSÕES

A assistente pessoal, Lana, pode ser vista em funcionamento no Messenger e no Telegram na Figura 5A, com a assistente pessoal utilizando o Messenger. Pode ser observado que é perguntado ao usuário se ele já utilizou a Lana anteriormente, o usuário responde que sim e então é requisitado ao usuário as credenciais que haviam sido utilizadas no cadastro com a assistente.

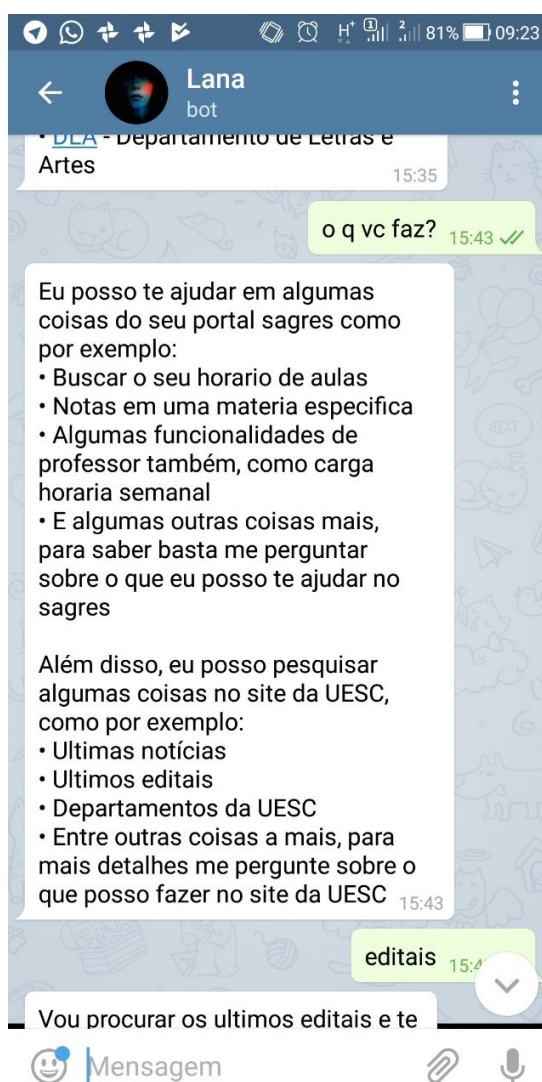
Figura 5 – Primeira conversação com a Lana e serviços ofertados.



Fonte: elaborado pelo autor.

Já a Figura 5B apresenta uma saudação, uma conversa trivial, e em seguida uma pergunta de um usuário que deseja saber em que a Lana pode ajudá-lo, i.e., quais são os serviços disponíveis pela assistente pessoal. A mesma pergunta pode ser realizada de maneira diferente, como é mostrado na Figura 6, onde é apresentado um cliente perguntando a Lana, a partir do Telegram, quais são os serviços realizados por ela, então é respondido de maneira geral algumas das principais ações implementadas. Nota-se que apesar da mensagem conter gírias como “q” e “vc” a Lana ainda consegue extrair a intenção do usuário de saber quais serviços ela pode realizar.

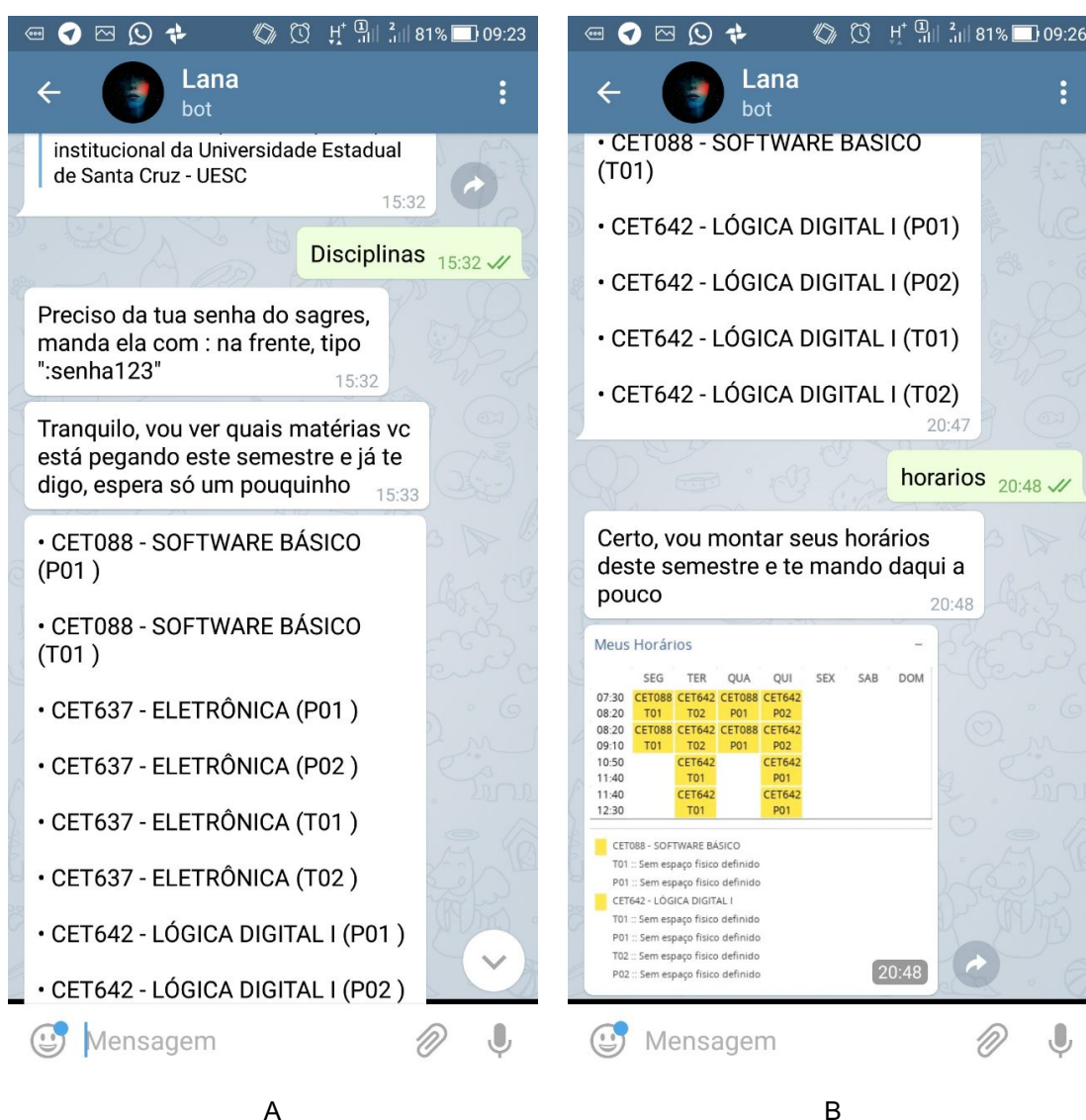
Figura 6 - Lana informando suas principais funcionalidades.



Fonte: elaborado pelo autor.

A Figura 7 apresenta a Lana realizando ações para um professor por meio da aplicação Telegram, na Figura 7A é possível observar a requisição da listagem de disciplinas ministradas com uma mensagem mais direta, somente falando “Disciplinas” e ainda assim foi possível extrair a intenção, em seguida a AP requisita ao cliente a senha do portal Sagres para realizar tal operação, visto que este usuário já havia utilizado a Lana posteriormente não foi necessário informar o nome de usuário, após receber a senha foi realizado o serviço e enviado a resposta ao cliente.

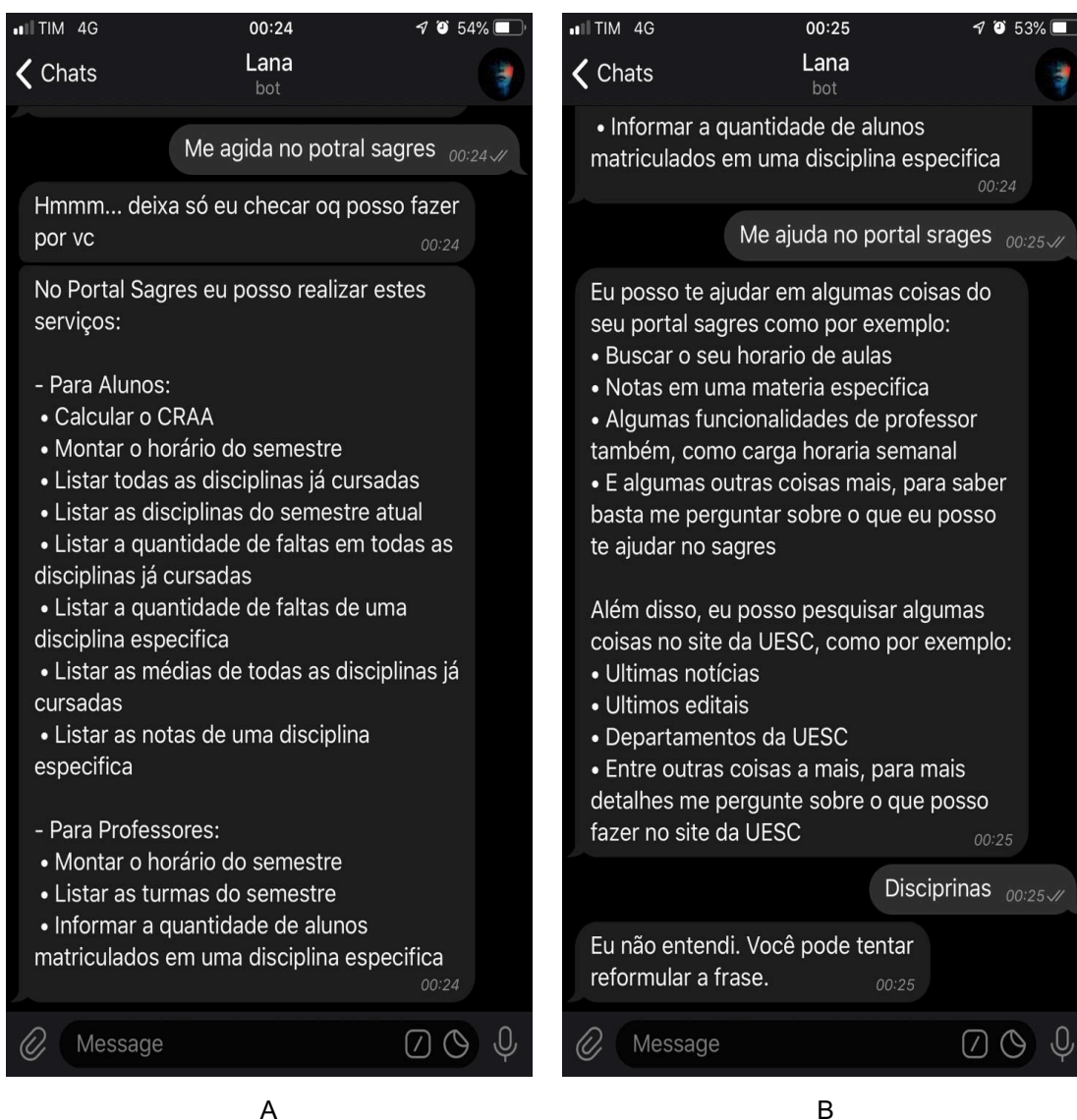
Figura 7 - Lana realizando ações para professor no Telegram.



Fonte: elaborado pelo autor.

Em seguida a Figura 7B apresenta a realização de uma outra requisição para o envio dos horários de aulas a serem ministrados no semestre, por fim a Lana responde a essa requisição enviando uma imagem contendo estes horários. Também se observa que foi possível extrair a intenção de buscar os horários mesmo numa mensagem mais direta.

Figura 8 - Interpretações com erros de digitação.



Fonte: Elaborado pelo autor.

A Figura 8 mostra algumas interpretações feitas pela Lana a partir de mensagens com erros de digitação. A Figura 8A apresenta um usuário enviando uma mensagem com as palavras “ajuda” e “portal” com erros de digitação e mesmo assim

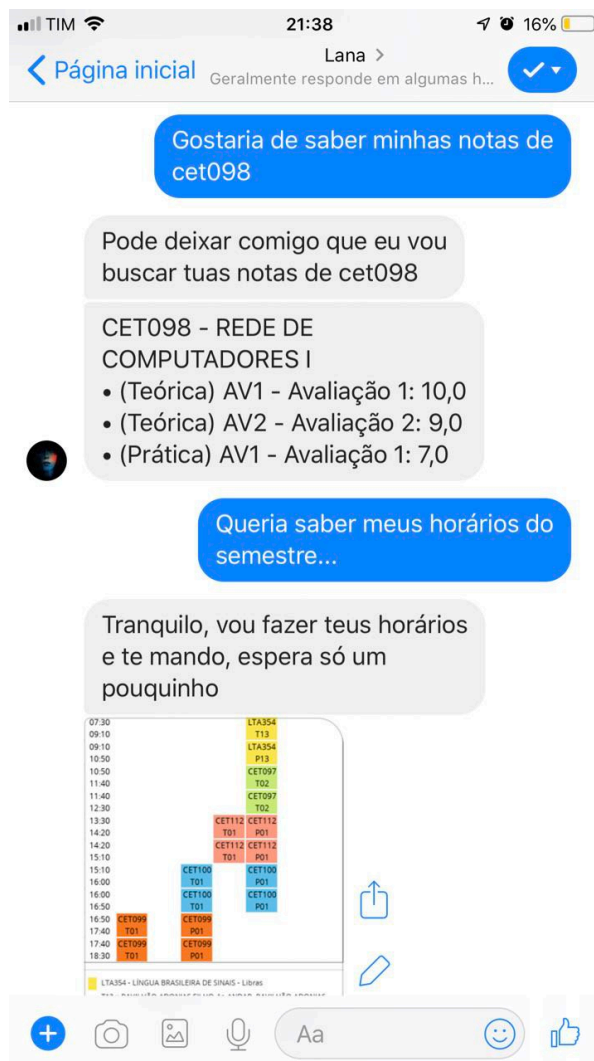
foi possível extrair a intenção de buscar os serviços que podem ser realizados no portal sagres.

Entretanto nem sempre é possível extrair a intenção corretamente quando ocorrem erros de digitação, a Figura 8B ilustra um outro erro de digitação similar, onde o usuário escreve “sagres” de maneira incorreta, nesta a Lana extrai a intenção incorreta, é respondido ao usuário as ações gerais que podem ser realizadas pela assistente e não as específicas do portal sagres como havia sido requisitado. Ainda na Figura 8B, também é enviado uma mensagem com erro de digitação com o objetivo de requisitar a ação de listar as disciplinas no portal sagres, neste caso não foi possível identificar nenhuma intenção na mensagem, sendo assim a assistente pessoal responde ao usuário que não conseguiu entender, o mesmo tipo de mensagem também é enviado ao usuário quando não é possível identificar nenhuma das intenções cadastradas.

Na Figura 9, podemos observar o funcionamento da Lana no aplicativo de troca de mensagens Messenger, nesta é apresentado duas requisições de serviço à Lana, ambas ações realizadas para um aluno no portal acadêmico sagres, uma para a busca de notas da disciplina de código “CET098”, onde é retornado ao usuário a lista de notas referentes as avaliações desta matéria. Em seguida, outra requisição para que a assistente busque os horários de aula no portal Sagres, nesta a Lana envia como resposta uma imagem que contem o horário de aulas semanais do requisitante. Pode-se notar que a código da disciplina foi extraído da mensagem em conjunto a intenção, desta maneira o AP já conseguiu identificar que deveria executar um serviço para o código em questão, além disso, a Figura também mostra que é possível requisitar o mesmo serviço de maneiras diferentes, assim como na Figura 7B, é requisitado a Lana os horários do semestre, entretanto na Figura 9 o pedido foi realizado diferente, de maneira mais contextualizada.

Apesar do agente de interface do Messenger ter sido implementado, este só pode ser utilizado pelo criador da página do Facebook, para liberar o acesso ao público é necessário enviar uma requisição de análise ao Facebook, e somente após esta análise e liberação é concedido o uso do *bot* do Messenger ao público geral. Por fim, o desenvolvimento dos agentes de interface foi realizado assim como o esperado, sem ocorrer nenhum problema.

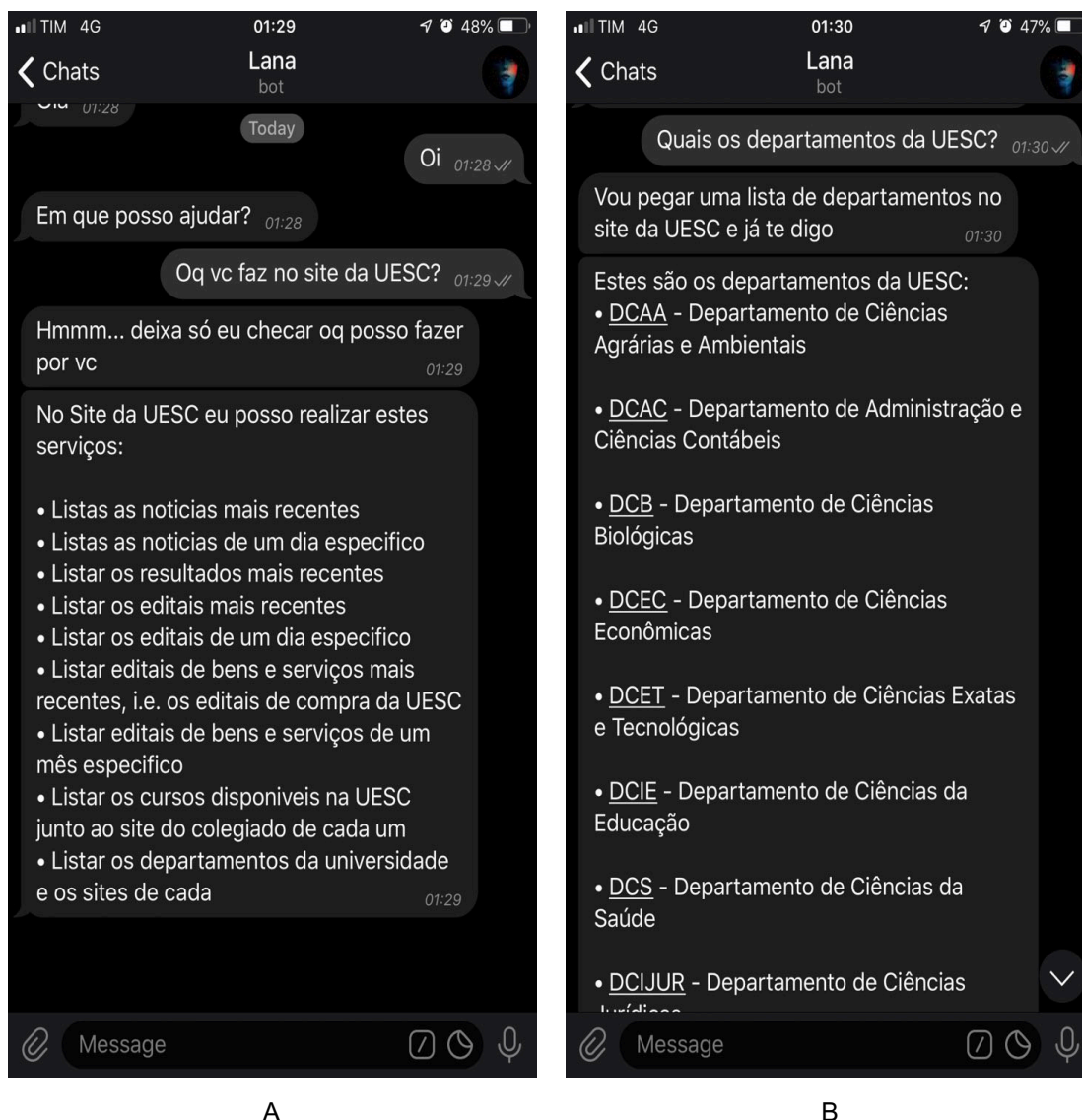
Figura 9 – Ações para aluno no portal acadêmico sagres no aplicativo Messenger.



Fonte: elaborado pelo autor.

Algumas ações realizadas pelo BotUESC podem ser observadas na Figura 10. A Figura 10A apresenta um usuário enviando uma mensagem com a intenção de listar as informações que possam ser recuperadas do site da UESC, em seguida a Lana responde quais são as ações possíveis no site da universidade. Já a Figura 10B apresenta uma requisição de listagem de departamentos da instituição, esta intenção também é extraída com sucesso, o serviço é executado sem problemas e então é enviado ao usuário uma lista com os departamentos da UESC, nota-se que esta lista possui marcação de estilo, onde as siglas de cada departamento possuem *hyperlinks* para os sites dos departamentos em si.

Figura 10 - Requisição de ações do BotUESC no Telegram.



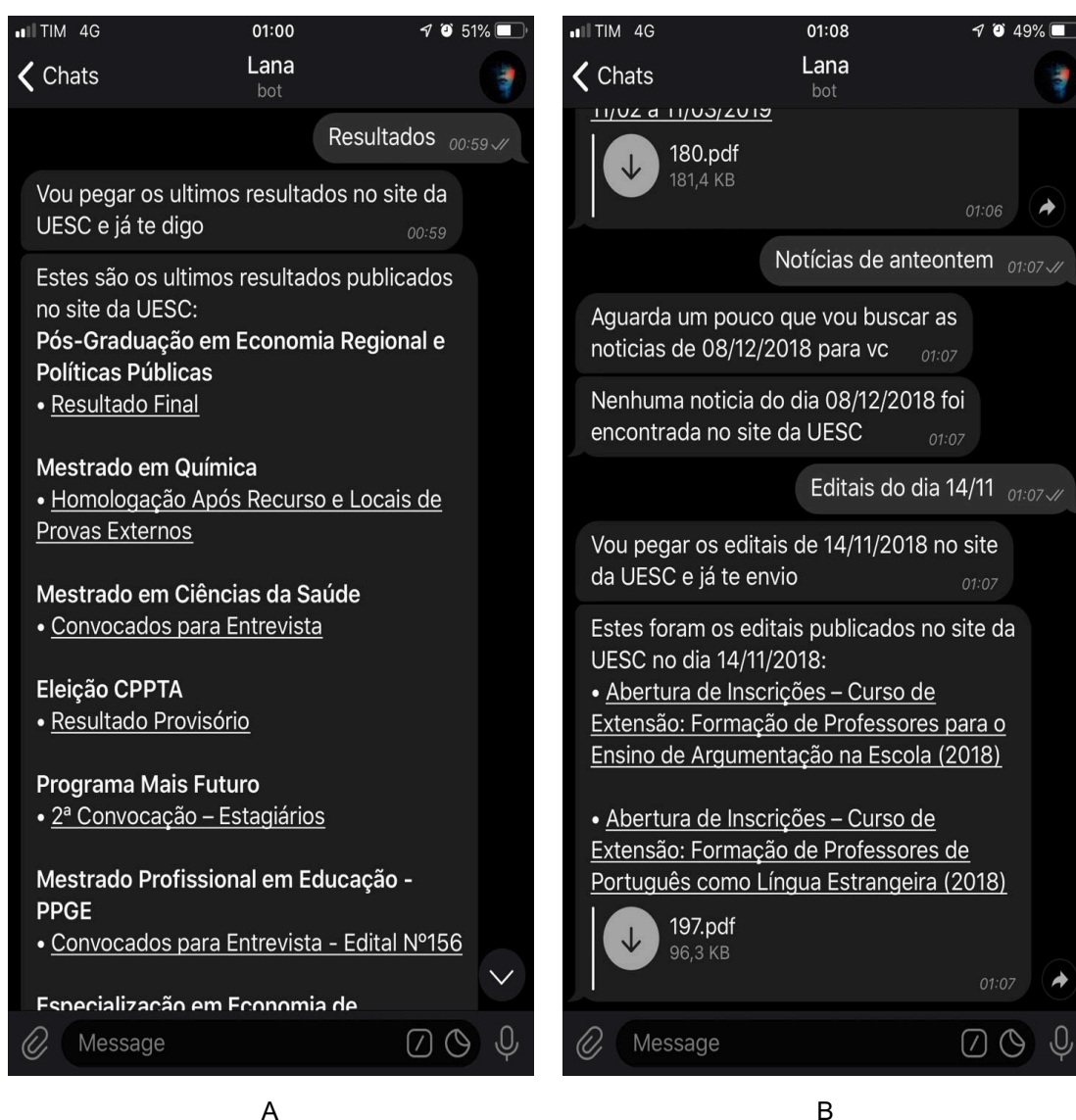
Fonte: Elaborado pelo autor.

A Figura 11 ilustra mais ações implementadas no BotUESC em funcionamento. Na Figura 11A é requisitado os últimos resultados publicados no site da universidade, a mensagem enviada, apesar de conter somente a palavra “resultados”, consegue ser interpretada corretamente e por fim o serviço é executado corretamente, enviando como resposta ao usuário a lista contendo os últimos resultados publicados no site da UESC juntamente aos links para o acesso direto de cada item encontrado.

Já a Figura 11B mostra uma mensagem enviada no dia 10 de dez. 2018 requisitando “Notícias de anteontem”, mesmo sem deixar explícito uma data foi possível identificar que o dia em questão seria 8 de dez. 2018, então é buscado no

site da universidade as notícias publicadas neste dia, entretanto nenhuma notícia havia sido publicada no dia em questão, sendo assim, a Lana avisa ao usuário que não foi encontrada nenhuma notícia naquele dia. Em seguida, ainda na Figura 11B é requisitado os editais publicados no dia 14 de nov. 2018, a intenção é processada corretamente e o serviço é executado sem maiores problemas, então é respondido ao usuário quais foram os editais publicados neste dia junto ao arquivo PDF de cada um deles.

Figura 11 - Algumas ações realizadas no site da UESC no Telegram.



Fonte: Elaborado pelo autor.

A utilização da plataforma BaaS Back4app acelerou a implementação do módulo de banco de dados, visto que não foi necessária criar a infraestrutura de um banco de dados, assim como não foi preciso criar uma API para as operações básicas do banco. Mesmo sendo necessário o estudo de novas ferramentas para o uso da plataforma, como a biblioteca do Parse, somente foi preciso estudar a documentação disponibilizada pelo Back4app para a criação das funcionalidades deste módulo.

O Watson Assistant realizou todo o trabalho de entendimento de linguagem natural. A interface oferecida pela IBM para o uso do serviço é intuitiva e possui documentação explicando todos os conceitos necessários para o uso do Watson Assistant, o que ajudou na configuração do serviço para o uso no módulo de entendimento de linguagem natural. Apesar do problema na criação de *Dialogs* de *Intents* do BotSagres apresentado no tópico 3.4.1, um maior estudo do Watson Assistant poderia ajudar a resolver estes mesmo problemas com uma abordagem diferente sem a necessidade de repassar a competência de resolução a outro módulo.

Assim como planejado, o módulo de assistente pessoal foi totalmente construído para realizar grande parte das suas ações de forma genérica, sendo as únicas ações específicas as de realizar operações com as informações dos usuários, como remover, salvar ou alterar, entretanto essas funcionalidades não são específicas a variáveis e informações pré-determinadas, mas sim de forma dinâmica, bastando seguir aos padrões de comunicação da IDL.

Durante o desenvolvimento do BotUESC foram enfrentados alguns problemas e empecilhos gerados pela má estruturação HTML e falta de padronização apresentada pelo *site* da universidade. Algumas funcionalidades pensadas foram descartadas durante a análise de viabilidade devido à falta de organização das informações. Por exemplo, uma ação que viria a ser implementada seria a de encontrar o nome, localização no campus e currículo dos discentes de cada curso. Entretanto, esta não é viável pois o *site* não possui nenhuma padronização para dispor estas informações, alguns cursos disponibilizam estes dados em forma de tabela, outros em arquivos do tipo PDF e até mesmo nomes diferentes no menu de opções que possuem a mesma funcionalidade. Estas incoerências na disposição dos dados no site dificultam a automatização da recuperação de informações, pois seria necessário implementar diferentes maneiras de buscar dados semelhantes para cada curso e departamento da universidade, o que deixaria o código fonte mais complexo e dificultaria a manutenção do *bot* caso houvesse novas mudanças na estruturação

das páginas. A dificuldade e extensão de implementações específicas foram os motivos para a inviabilização desta e de outras mais ações. Por fim, todas as funcionalidades propostas foram implementadas sem problemas e obtiveram resultados corretos, como já esperado, visto que parte das funcionalidades desejadas já haviam sido consideradas inviáveis.

Já o desenvolvimento do BotSagres foi realizado sem empecilhos, todas as funcionalidades desenvolvidas foram testadas e obtiveram resultados corretos. Somente um problema foi identificado durante a implementação deste extrator de dados. A comunicação entre todos dos módulos do SAP é realizada de maneira criptografada utilizando o protocolo de comunicação HTTPS e todas as informações de usuários armazenadas são criptografadas, entretanto o *síte* do portal Sagres não implementa nenhum tipo de segurança com criptografia das informações de autenticação para acesso ao sistema, toda a sua comunicação é realizada por meio do protocolo HTTP, o que gera uma falha de segurança expondo os nomes de usuário e senhas de alunos e professores para a rede ao acessar o sistema.

5 CONCLUSÃO

Neste trabalho foi proposto a implementação de um *software* de assistência pessoal distribuído capaz de receber mensagens textuais, processar linguagem natural a fim de extrair as intenções a respeito dos serviços propostos e recuperar informações do portal acadêmico sagres e do site da UESC.

O módulo de interface implementado apresentou a capacidade de estabelecer a comunicação entre o usuário e o módulo de assistente pessoal. Já o armazenamento de dados dos usuários foi tratado no módulo de banco de dados, o módulo de entendimento de linguagem se mostrou capaz de identificar as intenções relacionadas aos serviços dos extratores de dados e algumas intenções triviais. O BotSagres se mostrou capaz de recuperar informações no portal acadêmico sagres, o BotUESC também se mostrou funcional para buscar dados no site da UESC. Por fim, o módulo de assistente pessoal foi responsável por orquestrar a comunicação entre os outros módulos, criando assim um sistema distribuído de assistência pessoal capaz de receber mensagens de texto, interpretá-las e realizar as ações necessárias para responder ao usuário final, alcançando assim todos os objetivos propostos para este trabalho.

5.1 Trabalhos Futuros

Ao longo do desenvolvimento do sistema, pôde-se identificar possíveis melhorias a serem implementadas em trabalhos futuros, tais como: melhoramento do entendimento de linguagem natural; implementação de novas conversações, implementar conversação a partir de mensagens de áudio visando melhorar a acessibilidade do SA; criação de novos serviços para o SAP, assim como a criação de novos indexadores de *bots* com temas diferentes; calcular eficiência em tempo de resposta do assistente e a implementação de testes automatizados que sejam realizados periodicamente para a identificação de erros nos serviços visando maior rapidez na correção de novos problemas.

REFERÊNCIAS

7GRAUS. **Significado de Kanban**: o que é, conceito e definição. Disponível em: <<https://www.significados.com.br/kanban/>>. Acesso em: 8 nov. 2018.

BATSCHINSKI, George. **What is a Backend as a Service?** Disponível em: <<https://blog.back4app.com/2016/01/11/what-is-a-backend-as-a-service/#more-705>>. Acesso em: 8 nov. 2018.

BECK, K et al. **Agile Manifesto**. The Agile Manifesto, 2001.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Distributed Systems: Concepts and Design**, 5 ed. [S.l: s.n.], 2012.

DIGITALOCEAN. **DigitalOcean Droplets**. Disponível em: <<https://www.digitalocean.com/products/droplets/>>. Acesso em: 8 nov. 2018.

DIGITALOCEAN. **Droplet Overview**: DigitalOcean Product Documentation. Disponível em: <<https://www.digitalocean.com/docs/droplets/overview/>>. Acesso em: 20 nov. 2018.

ENEMBRECK, F.; BARTHES, J.-P. **Personal assistant to improve CSCW**. The 7th International Conference on Computer Supported Cooperative Work in Design, p. 329–335, 2002.

FERRUCCI, D. A. **Introduction to “This is Watson”**. IBM Journal of Research and Development, 2012.

FLANAGAN, David. **JavaScript: The Definitive Guide**, 6 ed. [S.l: s.n.], 2011.

GONZALEZ, Marco; LIMA, Vera L. S. De. **Recuperação de Informação e Processamento da Linguagem Natural**. XXIII Congresso da Sociedade Brasileira de Computação. Anais da III Jornada de Mini-Cursos de Inteligência Artificial. Campinas:[sn], v. 3, p. 347–395, 2003. Disponível em: <http://www.erfelipe.com.br/artigos/RI_Processamento_de_linguagem_natural.pdf>. Acesso em: 5 de dez. 2018.

GOOGLE. **DialogFlow - Docs**. Disponível em: <<https://dialogflow.com/docs>>. Acesso em: 5 dez. 2018.

GRIFFIN, Keith; FLANAGAN, Colin. **Defining a call control interface for browser-based integrations using representational state transfer**. Computer Communications, 2011.

HAHN, Rodrigo Machado; BARBOSA, Jorge Luis Victória. **Uma Arquitetura de Assistente Pessoal Orientada a Ambientes de Aprendizagem Ubíqua**. Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE), 2008.

HEROKU. **The Heroku Platform**. Disponível em: <<https://www.heroku.com/platform>>. Acesso em: 8 nov. 2018.

HOLMES, Antawan; KELLOGG, Marc. **Automating functional tests using selenium**. [S.l: s.n.], 2006.

IBM. **IBM Cloud**. Disponível em: <<https://console.bluemix.net/docs/overview/ibm-cloud.html#overview>>. Acesso em: 8 nov. 2018.

IBM. **IBM Knowledge Center - What is distributed computing**. Disponível em: <https://www.ibm.com/support/knowledgecenter/en/SSAL2T_8.2.0/com.ibm.cics.tx.doc/concepts/c_wht_is_distsd_comptg.html>. Acesso em: 19 nov. 2018.

KISER, Matt. **Introduction to Natural Language Processing (NLP) | Algorithmia Blog**. Disponível em: <<https://blog.algorithmia.com/introduction-natural-language-processing-nlp/>>. Acesso em: 9 dez. 2018.

LANE, Kin. **Overview Of The Backend as a Service (BaaS) Space**. API Evangelist, v. 2013, n. May, 2013. Disponível em: <<https://s3.amazonaws.com/kinlane-productions/whitepapers/API+Evangelist+-+Overview+of+the+Backend+as+a+Service+Space.pdf>>. Acesso em: 5 de dez. 2018.

MALIK, Sanjay Kumar; RIZVI, Sam. **Information extraction using web usage mining, web scrapping and semantic annotation**. 2011, [S.l: s.n.], 2011.

MARK, Massé. **REST API Design Rulebook**. [S.l: s.n.], 2013.

MESSERSCHMITT, David G. **Interface Definition Language**. University of California, Oakland CA, 1999.

MILLER, M. **IBM Cloud Docs Conversation**. Disponível em: <<https://console.bluemix.net/docs/services/conversation/index.html#about>>. Acesso em: 8 nov. 2018.

MITCHELL, Tom M. et al. **Experience with a learning personal assistant**. Communications of the ACM, 1994.

MOSTAÇO, Gustavo Marques et al. **AgronomoBot: a smart answering Chatbot applied to agricultural sensor networks**. 2018, [S.l: s.n.], 2018.

PAULO. **Top 10 Best Web Scraping Books - Simplified Web Scraping**. Disponível em: <https://nocodewebscraping.com/top-10-web-scraping-books/#What_Is_Web_Scraping>. Acesso em: 6 dez. 2018.

PERNA, CL; DELGADO, HK; FINATTO, MJ. **Linguagens especializadas em corpora: modos de dizer e interfaces de pesquisa**. [S.l: s.n.], 2010. Disponível em: < <https://goo.gl/fLRoyk> >. Acesso em: 10 de dez. 2018.

PITON, Otávio Henrique Gotardo. **AUTOMAÇÃO RESIDENCIAL UTILIZANDO A PLATAFORMA EM NUVEM IBM BLUEMIX**. 2017.

PYTHON. **What is Python? Executive Summary**. Disponível em: <<https://www.python.org/doc/essays/blurb/>>. Acesso em: 10 dez. 2018.

REATEGUI, Eliseo; RIBEIRO, Alexandre; BOFF, Elisa. **Um Sistema Multiagente Para Controle De Um Assistente Pessoal Aplicado a Um Ambiente Virtual De Aprendizagem**. Renote, 2008.

SOARES, MICHEL DOS SANTOS. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**. Idesia (Arica), v. 3, p. 6, 2004. Disponível em: <<http://www.dcc.ufpa.br/infocomp/index.php/INFOCOMP/article/view/68>>. Acesso em: 5 de dez. 2018.

SOEDIONO, Budi. **Web Scraping with Python**. [S.l: s.n.], 1989.

SOMMERVILLE, Ian. **Engenharia de Software**. [S.l: s.n.], 2013.

SUTIKNO, Tole et al. **WhatsApp, Viber and Telegram which is Best for Instant Messaging?** International Journal of Electrical and Computer Engineering (IJECE), v. 6, n. 3, p. 909, 1 jun. 2016.

TANENBAUM, Andrew S.; VAN STEEN, Maarten. **Distributed Systems: Principles and Paradigms**. [S.l: s.n.], 2013.

TELENYK, Sergii et al. **MODELS AND METHODS OF RESOURCE MANAGEMENT FOR VPS HOSTING**. 2013.

USACHEV, Denis et al. **Open source platform Digital Personal Assistant**. 2018. Disponível em: <<http://arxiv.org/abs/1801.03650>>. Acesso em: 19 nov. 18.

VARGIU, Eloisa; URRU, Mirko. **Exploiting web scraping in a collaborative filtering- based approach to web advertising**. Artificial Intelligence Research, v. 2, n. 1, 20 nov. 2012.

VENTRON. **METODOLOGIAS ÁGEIS: O QUE SÃO E PRINCIPAIS TIPOS**. Disponível em: <<https://www.ventron.com.br/blog/metodologias-ageis>>. Acesso em: 6 dez. 2018.

WOOLDRIDGE, Michael. **Introduction to Multiagent Systems**. Information Retrieval, 2002.

YAN, Mengting et al. **Building a Chatbot with Serverless Computing**. 2016, [S.l: s.n.], 2016.

ZAMBIASI, Saulo Popov; RABELO, Ricardo J. **Uma Arquitetura Aberta e Orientada a Serviços para Softwares Assistentes Pessoais**. Revista de Informática Teórica e Aplicada, 2012.