UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

BotAnalyzer: Uma ferramenta para reconhecer as interações entre aluno e chatbot

Samuel de Oliveira Gamito

São Carlos – SP

BotAnalyzer: Uma ferramenta para reconhecer as interações entre aluno e chatbot

Samuel de Oliveira Gamito

Orientador: **Prof. Dra. Simone do Rocio Senger de Souza**

Coorientador: **Ms. Leo Natan Paschoal**

Monografia final de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como requisito parcial para obtenção do título de Engenheiro em Engenharia de Computação.
Área de Concentração: Processamento de Linguagem Natural

USP – São Carlos
Junho de 2019

Gamito, Samuel de Oliveira
BotAnalyzer: Uma ferramenta para reconhecer as interações entre aluno e chatbot / Samuel de Oliveira Gamito. - São Carlos - SP, 2019.
68 p.; 29,7 cm.

Orientador: Simone do Rocio Senger de Souza.
Coorientador: Leo Natan Paschoal.
Monografia (Graduação) - Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos - SP, 2019.

1. Análise de Conversação. 2. Chatbot. 3. Processamento de Linguagem Natural. I. Souza, Simone do Rocio Senger de. II. Instituto de Ciências Matemáticas e de Computação (ICMC/USP). III. Título.

AGRADECIMENTOS

Agradeço a todas as pessoas que conheci durante meus anos de graduação, pois contribuíram para meu crescimento pessoal, acadêmico e profissional.

Agradeço aos meu pai e a minha mãe por toda ajuda e apoio que me deram durante todos os meus anos.

Um agradecimento especial ao Leo e a professora Simone por todo apoio e paciência durante o desenvolvimento desse projeto.

À República Várzea, por todos em que nunca estive sozinho.

“Quand nous passons trop de temps à voyage,
nous devenons des etrangers dans notre
propre pays.”
(René Descartes)

RESUMO

GAMITO, S. O. . **BotAnalyzer: Uma ferramenta para reconhecer as interações entre aluno e chatbot**. 2019. 68 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Os chatbots têm sido utilizados com sucesso para apoio em modelos pedagógicos como *Flipped Classroom*, mecanismo que auxilia estudantes, solucionando suas dúvidas durante o estudo. Apesar das oportunidades geradas pelos *chatbots*, atualmente, professores não tem a sua disposição recursos visuais e práticos que consigam indicar como foi a interação dos estudantes com o *chatbot*. Portanto, os professores não tem visibilidade das dúvidas mais comuns, dos assuntos mais difíceis de serem entendidos, das questões que o *chatbot* não conseguiu responder ou mesmo se os alunos realmente utilizaram o agente conversacional. Diante disso, este trabalho tem o propósito de definir uma ferramenta de apoio ao professor com informações das interações humano-*chatbot*. Foram estudados *frameworks* de processamento de linguagem natural, uma ferramenta foi especificada e desenvolvida (BotAnalyzer) e um estudo foi conduzido para compreender e explicitar o funcionamento da mesma.

Palavras-chave: Análise de Conversação, Chatbot, Processamento de Linguagem Natural.

ABSTRACT

GAMITO, S. O. . **BotAnalyzer: Uma ferramenta para reconhecer as interações entre aluno e chatbot**. 2019. 68 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Chatbots have been successfully used to support pedagogical models such as Flipped Classroom, a mechanism that helps students, solving their doubts during the study. Despite the opportunities generated by chatbots, currently, teachers do not have visual and practical resources to indicate how students interact with chatbot. Then, teachers have no visibility of the most common questions, the most difficult subjects to understand, the questions that the chatbot could not answer or even whether the students actually used the conversational agent. Therefore, this paper aims to define a tool to support teachers with information from human-chatbot interactions. Natural language processing frameworks were studied, a tool was specified and developed (BotAnalyzer) and a study was conducted to understand and explain its operation.

Key-words: Conversation Analysis, Chatbot, Natural Language Processing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização e Motivação

O termo *chatterbot* (ou chatbot) caracteriza programas de computadores capazes de interagir com usuários utilizando a linguagem natural (MAULDIN, 1994). Inicialmente esses programas foram desenvolvidos sem uma aplicação específica, apenas como prova de conceito. Eles utilizavam formas simplificadas de casamento de padrões para encontrar uma resposta adequada para as entradas do usuário (SHAWAR; ATWELL, 2007), como a *chatbot* ELIZA (WEIZENBAUM, 1966).

Os *chatbots* vêm sendo utilizados em inúmeras áreas como: Serviços de atendimento ao cliente (SAC), em sistemas de vendas online, como ferramenta para auxílio nos estudos, entre outras (BRAUN, 2013). Esse aumento na utilização dos agentes conversacionais pode estar relacionado com o avanço das técnicas de processamento de texto, surgimento de *Corpora*, advento de anotações de linguagens mais robustas, como a AIML, possibilitou a eclosão de novos *chatbots* que fossem mais práticos e comerciais

No contexto educacional, os *chatbots* vem sendo utilizados como ferramenta para auxílio do aluno durante o estudo (PASCHOAL *et al.*, 2016; PASCHOAL; CHICON; FALKEMBACH, 2017; PASCHOAL; SOUZA, 2018). Seja na forma de tutores (MASSARO *et al.*, 2006), solucionando dúvidas e evitando equívocos durante os estudos (PASCHOAL; SOUZA, 2018). Também são capazes de fornecer material multimídia sem interferir ou oferecer danos cognitivos ao aluno, já que não conseguem gerar sobrecarga cognitiva no aluno (SANTOS, 2009).

Os *chatbots* também estão sendo utilizados em diferentes modalidades de ensino (ALENCAR; NETTO, 2010). Atualmente, muitos são os esforços para utilizar o *chatbot* como tutor em ambientes de educação online, como por exemplo *Massive Open Online Courses* (MOOC) (BOLLWEG *et al.*, 2018), já que os cursos não possuem professores disponíveis 24 horas por dia. Em outro seguimento de ensino, mais especificamente em práticas de educação presencial (LEONHARDT *et al.*, 2003; PASCHOAL, 2019), os *chatbots* podem auxiliar o aluno durante o treinamento de um conteúdo. Apesar de aliviar a carga de trabalho do professor, criando um ambiente 24 horas onde os alunos possam resolver suas dúvidas, poucos são os esforços durante o processos desenvolvimento dos agentes conversacionais pensando em diminuir a lacuna criada pelos *chatbots* na comunicação entre o professor e o aluno. Os poucos casos de ferramentas

que visam auxiliar o professor, são ferramentas que ajudam na criação de *chatbots* e não na visualização das interações que ocorreram nos *chatbots* (KRASSMANN *et al.*, 2017).

Com o auxílio dos *chatbots* o professor pode implantar modelos pedagógicos que exigem do aluno o estudo prévio sobre o conteúdo teórico (atividade passiva), já que essa ferramenta pode aliviar a carga do professor resolvendo problemas triviais que os alunos possam ter durante os estudos. Entretanto, a única forma de *feedback* que o professor tem destas interações são conjuntos de *logs* formatados com um padrão de entradas e saídas o que torna difícil identificar as dúvidas que são comuns aos alunos, os assuntos que são mais difíceis de serem entendidos, as dúvidas que perfazem a turma porque o *chatbots* não conseguiu solucioná-las, dentre outros.

Tendo em vista essa falta de recursos para *feedback* do uso de *chatbots*, este projeto tem o propósito de estudar e desenvolver uma ferramenta que contribua com o professor que utiliza *chatbot* como mecanismo de apoio em suas aulas. Assim, tem a finalidade de investigar métodos e procedimentos derivados de processamento de linguagem natural e definir um mecanismo capaz de analisar os registros da interação entre humano-*chatbot* com o intuito de fornecer ao professor uma interface gráfica simples e amigável (*dashboard*), para que ele possa utilizar e ficar ciente sobre quais conteúdos geram mais dúvidas, quais as dúvidas dos alunos que não foram solucionadas, a quantidade de alunos que interagiu com o *chatbots*, o tempo médio de duração de seções de conversa da turma, dentre outras informações que poderão ser obtidas a partir da análise das conversações por meio de uma ferramenta automatizada.

1.2 Organização do Trabalho

Esta monografia está organizada da seguinte forma: O Capítulo 2 apresenta um embasamento teórico para que leitor possa ter os conhecimentos básicos utilizados no desenvolvimento da ferramenta, dentre eles destacam-se: formalização de linguagem natural, métodos computacionais para interpretação e processamento de texto, e formas para mineração de texto. Já o Capítulo 3, descreve as atividades realizadas durante o desenvolvimento do projeto, desde os estudo das ferramentas, bibliotecas e *frameworks*, a forma como foram implementadas, decisões tomadas durante o projeto e alguns algoritmos cruciais para o funcionamento da ferramenta. Por último, no Capítulo 4 são apresentadas as conclusões do trabalho e o seu impacto.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

2.1 Considerações Iniciais

Este capítulo apresenta alguns conceitos sobre os agentes conversacionais na Seção 2.2, descrevendo os agentes conversacionais, contextualizando a utilização destes em diferentes contextos. Também descreve sobre os conceitos de processamento de linguagem natural (Seção 2.3).

2.2 Agentes Conversacionais

2.2.1 Visão histórica

Em 1950, o cientista Alan Turing publicou um artigo no qual ele questionava se as máquinas poderiam pensar. Dentro deste artigo o autor estipulou que isso aconteceria quando uma maquina pudesse se passar por uma pessoa e confundir um interrogador que estaria tentando descobrir o sexo das partes interrogadas, o teste recebeu o nome de Jogo da Imitação e posteriormente Teste de Turing. Turing acreditava que, em cerca de 50 anos, sería possível programar computadores que conseguissem jogar o Jogo da Imitação.

Em 1966 o cientista da computação, Joseph Weizenbaum, demonstrou uma aplicação capaz de interagir com um interrogador utilizando linguagem natural (WEIZENBAUM, 1966). A aplicação denominada Eliza é reconhecida como o primeiro *chatbot*, ela consistia principalmente de métodos de análise de frases e fragmentos, buscando as palavras-chaves nos textos e montando sentenças a partir destes fragmentos (AGASSI; WIEZENBAUM, 1976).

Para poder interagir com uma pessoa, os *chatbots* necessitam interpretar a pergunta e buscar a resposta em uma base de conhecimento pré-definida. Para tanto muitas pesquisas na área de inteligência artificial e processamento de linguagem natural vem sendo desenvolvidas para criar e aperfeiçoar ferramentas e arquiteturas (GALVAO *et al.*, 2004). Atualmente o mecanismo de casamento de padrões com linguagem de marcação (AIML) é um dos mais utilizados para construção de agentes conversacionais (NEVES; BARROS; HODGES, 2006).

O avanço de ferramentas *web*, como por exemplo *Software as a Service* (SaaS), e a grande variedade de implementações em código aberto permitiram a disponibilização dos *chatbots* como ferramentas online, criando a possibilidade de serem desenvolvidos para as mais diversas áreas

(RADZIWILL; BENTON, 2017). No contexto educacional os *chatbots* vem ganhando força como ferramenta de apoio à aprendizagem em especial na área de computação. Além de oferecer o fluxo de conversa baseado em textos, os *chatbots* trazem formas de comunicação não verbais (LIN *et al.*, 2013), por texto, áudio, gestos, animações, dentre outras.

2.2.2 Chatbots no Contexto Educacional

No âmbito educacional os *chatbots* são utilizados para apoiar as mais diversas áreas de ensino, seja ela Computação, Medicina, Educação Especial, Física, *et al*. Eles auxiliam o ensino por serem capazes de fornecer referencial teórico, apresentar exemplos básicos, apresentar conteúdo multimídia, além de fornecerem *links* para tutoriais (PASCHOAL, 2019).

Ao longo dos anos inúmero autores publicaram soluções de agentes conversacionais para suprir as necessidades das áreas em que estão inseridos, na computação destacam-se os *chatbot*, Doroty um *chatbot* específico para profissionais na área de gerenciamento de redes de computadores (LEONHARDT, 2005), Cocorote tem o papel de tutor virtual no ensino de Algoritmos(LEMOS, 2011), Ubibot que foi projetado para apoiar o ensino da disciplina de Engenharia de Software (PASCHOAL *et al.*, 2016) e por fim temos o TOB-STT construído para auxiliar no aprendizado de Teste de Software (PASCHOAL *et al.*, 2019).

2.3 Processamento de Linguagem Natural

2.3.1 Conceitos sobre Processamento de Linguagem Natural

Diferentemente das ferramentas de processamento de dados, as ferramenta de processamento textual necessitam ter conhecimento da linguagem. Como exemplo o comando *wc* do Unix, enquanto utilizado para contar bytes e linhas ele atua como ferramenta de processamento de dados já que recebe uma entrada e sem conhecimento prévio consegue tratá-la, e assim emitir uma saída. Porém quando utilizado para contar palavras o comando *wc* precisa saber diferenciar as palavras dos outros conjuntos de símbolos, sendo assim, necessita conhecer previamente a linguagem analisada (JURAFSKY, 2018).

Conforme Covington, Nute e Vellino (1996) os esforços na área de processamento de linguagem natural estão nos três aspectos da língua:

- Forma (Morfologia): Consiste em analisar as classes gramáticas que constituem uma frase, no caso da língua portuguesa: substantivo, artigo, adjetivo, numeral, pronome, verbo, advérbio, preposição, conjunção e interjeição.
- Som (Fonologia): Consiste em analisar os fonemas que constituem as frases;
- Estrutura (Sintaxe): Estuda como definir a estrutura de uma frase com base na forma que as palavras se relacionam;

- Significado (Semântica): Busca associar os significados dos termos encontradas na sintaxe;

- Contexto (Pragmática): Estuda o significado das palavras e das frases dentro do contexto em que estão inseridas (MORATO; KOCH *et al.*, 2011).

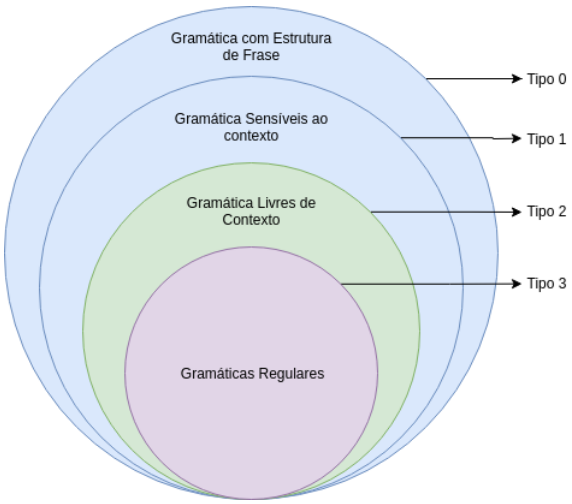
Para que seja possível extrair informações de textos não formatos que utilizam linguagem natural, utilizando métodos computacionais, é necessário definir formalmente a gramática que gera esses textos.

2.3.2 Definição Formal da Linguagem

Um alfabeto ou vocabulário é formalmente definido como um conjunto finito não vazio de símbolos, conhecidos individualmente como letras, porém nem todos os símbolos são exclusivamente letras, eles podem ser representados por números ou dígitos. Uma cadeia de símbolos é denominado palavra ou *string*, e uma linguagem é definida como uma coleção de cadeias de símbolos, dentro da linguagem essas cadeias são denominadas de sentenças de linguagem (HARRISON, 1978) Por exemplo, podemos definir o alfabeto binário como $\Sigma = \{0, 1\}$, sendo assim, um número binário é representado por uma sequência finita dos símbolos de Σ (CLARK; LAPPIN, 2015).

Tendo em vista a possibilidade de criar inúmeras gramáticas, Chomsky (1959) classifica as gramáticas formais em quatro níveis, são eles: Gramáticas com estrutura de frase (tipo 0), gramáticas sensíveis ao contexto (tipo 1), gramática livre de contexto (tipo 2), gramática regulares (tipo 3). A Figura 1 exemplifica por meio de diagrama de Venn a forma como esses níveis estão agrupados. Cada um dos níveis serão descrito a seguir.

Figura 1 – Hierarquia de Chomsky



Fonte: Elaborada pelo autor.

- **Gramáticas Regulares:** representam a forma mais restrita de gramática, descrevendo apenas linguagens regulares, que podem ser capturadas por expressões regulares.

  A seguir é apresentado um exemplo de gramática que gera todas as cadeias sobre o alfabetos compostos pelas as letras a e b, sendo a menor *ab* a menor gramática gerada.

$$S \to aS$$
$$S \to bA$$
$$A \to a$$

  As regras de criação a cima definem uma gramática de dois não-terminais $N = \{S, A\}$, três terminas $\sigma = \{a, b, c\}$ e descrevem a mesma linguagem da expressão regular $a*bc*$.

- **Gramáticas Livres de Contexto:** descrevem uma linguagem através de regras simples que podem cair em uma recursão, sendo assim não ficam presas a uma forma de reincidência como as gramáticas regulares. Utilizando o exemplo anterior, porém alterando as regras de criação, podemos criar uma regra de construção, que também utiliza as o alfabeto a e b, porém nesse exemplo é possível notar que as menores cadeia são a ou b, e esse conjunto de regras consegue gerar um alfabeto maior:

$$S \to A$$
$$A \to aA$$
$$A \to Ab$$
$$A \to a$$
$$A \to b$$

  As gramáticas livres de contexto possuem como restrição o fato de que apenas o lado direto da "$\to$"pode conter símbolos não-terminais.

- **Gramáticas Sensíveis ao Contexto:** Nessa gramática, tantos os lados direitos e esquerdos das regras de produção podem conter símbolos não terminais. As gramáticas sensíveis ao contexto são mais gerais que as gramáticas livres de contexto, porém ainda não é possível interpreta-las com automatos.

Ex:

$$S \rightarrow aBC$$
$$S \rightarrow aSBC$$
$$CB \rightarrow BC$$
$$aB \rightarrow ab$$
$$bB \rightarrow bb$$
$$bC \rightarrow bc$$
$$cC \rightarrow cc$$

Neste exemplo a gramática possui não-terminais $N = \{A, B, C\}$, terminais $\sigma = \{a, b, c\}$ e as regras de criação definidas a cima, esse conjunto gera a linguagem $\{a^n b^n c^n : n \geq 1\}$

- **Gramáticas com Estrutura de Frase:** também conhecida como Gramática Irrestrita, sendo assim, qualquer terminal e não-terminal pode aparecer em qualquer posição dentro de uma produção.

2.3.2.1 Gramática para Linguagem Natural

Poderia-se definir a língua inglesa, por exemplo, através de uma gramática com estrutura de frase, já que este tipo construção consegue gerar todas as combinações que um alfabeto pode gerar. Porém por seria necessário exclusivamente de uma maquina de Turing para interpretar esta linguagem (HOPCROFT RAJEEV MOTWANI, 2001). Não é possível utilizar uma gramatica simplificada, como a gramática regular, para representar essa linguagem mas teríamos um problema já que a gramática não conseguiria representar todas as cadeias, nem todas as regras de formação existente em uma linguagem natural (CLARK; LAPPIN, 2015).

O autor Joshi (1987) define uma gramatica que está entre as sensíveis ao contexto e a livres de contexto, ele a denomina de *Mildly Context-Sensitive Languages*, a qual é descrita por (CLARK; LAPPIN, 2015) como:

1. Contém todas as linguagens livres de contexto;
2. Pode ser analisada em polinômios;
3. Considera as construções de linguagem que as linguagens livres de contexto não conseguem solucionar(ex: *cross-serial dependence*).

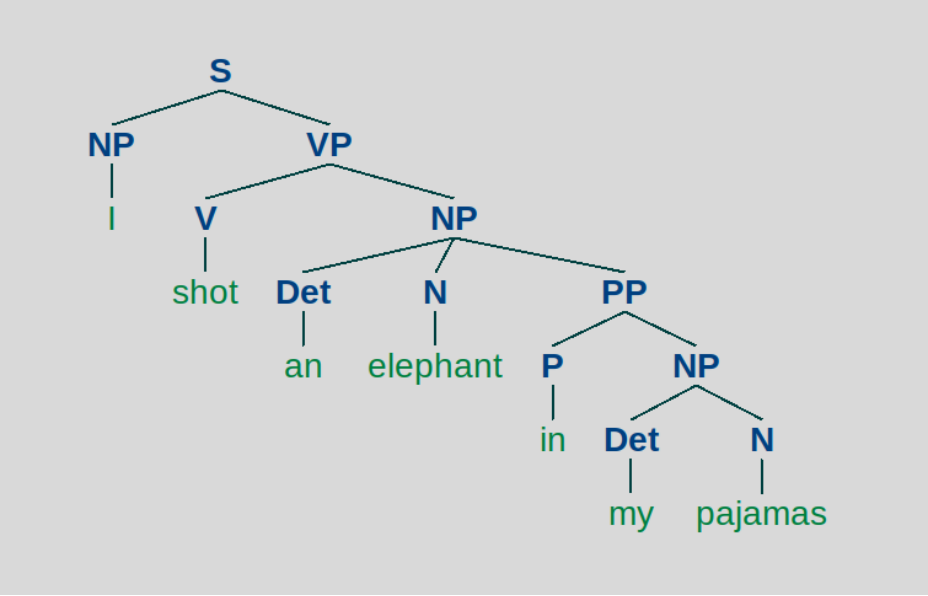
As *Tree Adjoining Grammars* (TAGs) (JOSHI; SCHABES, 1997), cumprem com os requisitos descritos a a linguagem *Mildly Context-Sensitive*. Enquanto as derivações de gramaticas livres de contexto estão limitadas a restrição do nó raiz, as TAGs permitem gerar nós

raízes dentro de um nó, pode-se dizer então que as derivações das TAGs não estão limitadas à concatenação de cadeia de characteres(GROUP *et al.*, 1998):

A Figura 2 exemplifica uma árvore gerada pela gramática a seguir,$N = \{NP, VP, P, NP\}$, $\Sigma = \{I, Shoot, an, elephant, in, my, pajamas\}$, e a regra de formação para a gramática e definida pelas seguintes relações:

$$
\begin{aligned}
S &\rightarrow NPVP \\
PP &\rightarrow PNP \\
NP &\rightarrow DetN|DetNPP|I \\
VP &\rightarrow VNP|VPPP \\
Det &\rightarrow an|my \\
N &\rightarrow elephant|pajamas \\
V &\rightarrow shot \\
P &\rightarrow in
\end{aligned}
$$

Figura 2 – TAG gerada a partir da frase "I shoot an elephant in my pajamas", utilizando a ferramenta NLTK



Fonte: Elaborada pelo autor.

Na figura 2 podemos ver graficamente a árvore de geração criada utilizando a biblioteca NLTK. Partindo de um nó raiz o algoritmo consegue separar interpretar e separar(*parsing* [1]) e a partir dessas regras pode

[1] É a tarefa de separa um texto em pedaços menores baseados em uma regra.

2.3.3 Análise Sintática

Utilizando a definição de linguagem feita na Seção 2.3.2 é possível escolher uma representação de linguagem que auxilie o computado separar e interpretar frases

Durante o processo de análise sintática o autor Hare *et al.* (2006) define duas abordagens para efetua-la: *Top-Down* e *Bottom-Up*. A primeira consiste em construir a linha de raciocino da a partir de um nó raiz e caminhar até os nós terminais (HAKEN, 2013). Já a segunda produz a árvore de derivação começando pelos terminais até chegar no nó raiz.

2.3.4 Análise Semântica

Para os humanos, entender um texto, uma frase ou uma palavra é quase que um processo inconsciente. Para entender uma frase uma pessoa conta com o conhecimento prévio da linguagem em que ela é escrita e dos conceitos contidos, porém um computador não é capaz de realizar esse tipo de análise.

Algumas técnicas fazem a máquina parecer entender um texto (RICHARDSON; BURGES; RENSHAW, 2013). Abordagens como análise estatística das palavras-chaves, ou até técnicas de aprendizado de máquina, casamento de padrões ou técnicas de análise de frequência para indicar o contexto do texto; essa técnicas tentam interpretar os textos entretanto não analisam o significado dos textos.

Sendo assim, a análise semântica surge como a finalidade de fazer um computador entender a forma como os humanos se comunicar através do contexto e do significado (RESNIK, 1995). Os métodos para isso se baseiam em identificar os termos mais importantes em uma sentença para entender sobre do que ela se trata(ANDREEVSKAIA; BERGLER, 2007).

2.3.4.1 Comparação Entre Palavras

Durante a atividade de processamento de linguagem natural, é necessário comparar trechos de textos ou palavras e determinar quão similares elas são. A literatura divide essas comparações em dois seguimentos (ANDREEVSKAIA; BERGLER, 2007):

- ***Corpus-based*:** Neste método o grau de similaridade associado as palavras derivam exclusivamente de grandes *corpora*. Por exemplo, é possível determinar o grau de similaridade entre as palavras olhando se elas costumam a aparecer nos mesmos textos e se elas aparecem próximas uma da outra e assim determinar a pontuação(TURNEY, 2001).

- ***Knowledge-based:*** Este método não se restringe ao processamento de linguagem natural (DALY *et al.*, 2002; SUBRAMANIAN *et al.*, 2005), ele é dividido em dua partes, base de conhecimento e mecanismo de inferência.

  A base de conhecimento é um conjunto de informações sobre um produto, área de comercio, ou palavras (MILLER, 1995).

Mecanismo de inferência tenta, a partir da base conhecimento, relacionar uma entrada, podendo ser uma palavra, com o conhecimento prévio.

Utilizando a *WordNet* (MILLER, 1995) como base de conhecimento, em que a relação das palavras são representadas por um grafo, podemos inferir a similaridade da duas palavras utilizando a equação 2.1 (LEACOCK; CHODOROW, 1998), onde *dist* é o menor caminho entre as palavras e P é a profundidade máxima da taxonomia da palavra na *WordNet*.

$$Sim = -log\frac{dist}{2*P} \tag{2.1}$$

2.3.4.2 Word Specificity

O *word specification* é uma medida estatística com a finalidade de indicar a importância de uma palavra em um texto. Geralmente ele é utilizado como peso para ponderar medidas em sistemas de busca(CHEN; SYCARA, 1998), modelagens (MIHALCEA *et al.*, 2006).

Segundo Jones (2004) podemos listar duas heurísticas para chegar à um valor que indique essa importância: *Term Frequency*(TF) e *Inverse Document Frequency*(IDF). O primeiro, TF, consiste em uma contagem de quantas vezes a palavra ou termo aparece no texto e a partir desta contagem atribuímos um valor a ele, esse valor pode ser o valor bruto, pode ser um valor binário que indica se ele esta ou não no texto, entre outras medidas. Já no segundo caso, IDF, indica quanta informação um termo provê. Para o seu cálculo é considerado em quantos *corpus* o termo aparece e quanto textos o *corpora*(representado pela letra D) possui, ele é dado pela fórmula. 2.2

$$IDF(t,D) = log\frac{tam(D)}{tam(t \in D)} \tag{2.2}$$

2.3.4.3 Semelhança Semântica de Texto

Tradicionalmente a semelhança semântica em textos é definida pela comparação entre as palavras contidas no textos. Para isso é utilizada uma métrica criada a partir contagem de vezes que essas palavras apareceram em recursos(*Corpus*) dentro de um mesmo contexto (MILLER, 1995), sendo assim elas podem ser consideradas sinônimos.

Não é possível fazer uma comparação entre textos ou conjuntos de duas ou mais palavras de uma forma direta(LEACOCK; CHODOROW, 1998). Entretanto, os autores Mihalcea *et al.* (2006) sugerem juntar técnicas de similaridade entre palavras e técnicas para determinar a *word specificity*(IDF) na frase para gerar uma nota que indique quão similar são as frases. Sendo assim, dados dois textos T_1 e T_2, para cada palavra p de T_1 buscamos uma palavra em T_2 com a maior similaridade entre palavras ($maxSim(p,T_2)$), feito isso fazemos uma soma ponderada desses valores com os valores de IDF, obtendo o grau de similaridade da frase T_1 com T_2. Repetindo o

processo contudo com o texto T_2 como base chegamos na equação 2.3 sugerida por Mihalcea *et al.* (2006).

$$sim(T_1, T_2) = 0,5 * (\frac{\sum_{p \in T_1} maxSim(p, T_2) * idf(p)}{\sum_{p \in T_1} idf(p)} + \frac{\sum_{p \in T_2} maxSim(p, T_1) * idf(p)}{\sum_{p \in T_2} idf(p)}) \quad (2.3)$$

2.4 Considerações Finais

Este capítulo trouxe uma breve história dos *chatbots* além de contextualizar a sua utilização em diversos cenários do dia a dia, além disso foram introduzidas formas de análise e processamento de texto que serão utilizadas para criar a ferramente de análise de *logs*.

No próximo capitulo é apresentando o trabalho desenvolvido, assim como os resultados obtidos, as decisões de projeto e as dificuldades e limitações encontradas em seu desenvolvimento

Capítulo 3

O BOTANALYZER

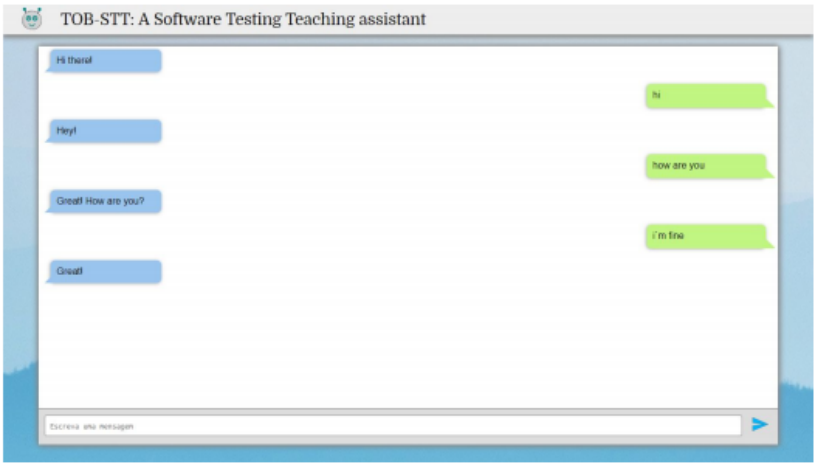
3.1 Considerações Iniciais

Este capítulo tem como objetivo apresentar o desenvolvimento, as decisões de projeto e a infraestrutura da ferramenta BotAnalyzer. A seção 3.2 descreve as funcionalidades da ferramenta e quais problemas a ferramenta visa solucionar. Já na seção 3.3 é apresentada a modelagem da ferramenta e as decisões que levaram para esse modelo.Na Seção 3.4 são apresentadas de forma detalhada as atividades realizadas ao longo do projeto para a criação da Api, enquanto isso a seção 3.5 apresenta as atividades feitas para a criação da interface gráfica. Fechando o capitulo, a seção 3.6 explica como a ferramenta é instalada e conFigurada.

3.2 Descrição da Ferramenta

Durante a interação de um usuário/aluno com um *chatbot*, o aluno deverá fazer perguntas ao *chatbot*, e após encontrar uma resposta o *chatbot* retorna a resposta, caso contrario o agente conversacional retorna uma resposta avisando que não foi possível responder aquela pergunta. Durante o processo de conversação o mecanismo do *chatbot* armazena todos os pares de perguntas e repostas em uma base de dados ou em um arquivo de *logs*, como exemplo de *chatbots* que apresentam esse funcionamento temos : Elektra(LEONHARDT *et al.*, 2003), Metis(KRASSMANN *et al.*, 2018), TOB-STT(PASCHOAL *et al.*, 2019), Ubibot(PASCHOAL *et al.*, 2016), dentre outros. Com a intenção de ilustrar, a Figura 3 apresenta um exemplo de interação de um aluno com o *chatbot* TOB-STT.

Figura 3 – Interface do *chatbot* TOB-STT



Fonte: Elaborada pelo autor.

O *chatbot* oferece um ambiente virtual disponível 24 horar por dia em que os alunos possam tirar suas dúvidas, e assim continuar com os seus estudos. Com isso os agentes conversacionais aliviam a carga de trabalho do professor. Em contra partida, o professor acaba perdendo contato com os alunos, gerando uma lacuna entre as dúvidas dos alunos possuem e o conhecimento dessas dúvidas pelo professor.

Para romper essa lacuna, o professor necessita fazer alguma análise de utilização do *chatbot*. E para isso ele necessita ler todas as interações que se encontram em uma base de dados ou arquivo de *logs*. E com a crescente utilização de *chatbot* em MOOC's (KRASSMANN *et al.*, 2017; PAZ *et al.*, 2017), um mecanismo de apoio para apoiar a análise de conversação se torna necessário, já que este mecanismos pode automatizar e gerar análises gráficas para o professor.

Se o professor resolver analisar a interação, ele irá se deparar com uma quantidade extensa de códigos, similar ao apresentado no trecho de Código 1. Nesse exemplo, tem-se dois pares de pergunta e resposta, na primeira o aluno pede para o *chatbot* um exemplo para o critério de particionamento de classe equivalência e na segunda interação o aluno apenas inicia a conversa com o agente conversacional. Visivelmente a forma padrão como ele é apresentado dificulta a leitura, uma vez que apresenta muitas informações impedindo uma visão global das interações. Além disso a análise desse *logs* pode exigir um conhecimento técnico do professor para que ele entenda o formato e o conteúdo desse arquivo.

Código-fonte 1: Exemplo de log do *chatbot* TOB-STT

```
1
2 [
3     {
4         "id":"988",
5         "input":"Can you give me an example of equivalence class
      partitioning criteria?",
```

```
6       "response":"Here is an example of an application of equivalence
        class partitioning. Consider the following specification: if you
        are below 18 years old, you are too young to be insured. If you
        are between 18 and 30 years old, you will get a 20% discount, but
        you are not elegible for an insurance if you are past 30. In this
        problem, the ages 18,.., 30 are in the same equivalence class",
7       "user_id":"63",
8       "convo_id":"dt8bn9943b2ssljfg272rg16j0",
9       "bot_id":"1",
10      "timestamp":"2019-02-28 12:10:50"
11    },
12    {
13      "id":"989",
14      "input":"TOB_STT iniciar conversa",
15      "response":"Hi there! What\\'s your name?",
16      "user_id":"63",
17      "convo_id":"dt8bn9943b2ssljfg272rg16j0",
18      "bot_id":"1",
19      "timestamp":"2019-02-28 21:36:17"
20    }
21 ]
```

A Tabela 1 apresenta uma descrição sobre cada campo do Código 1.

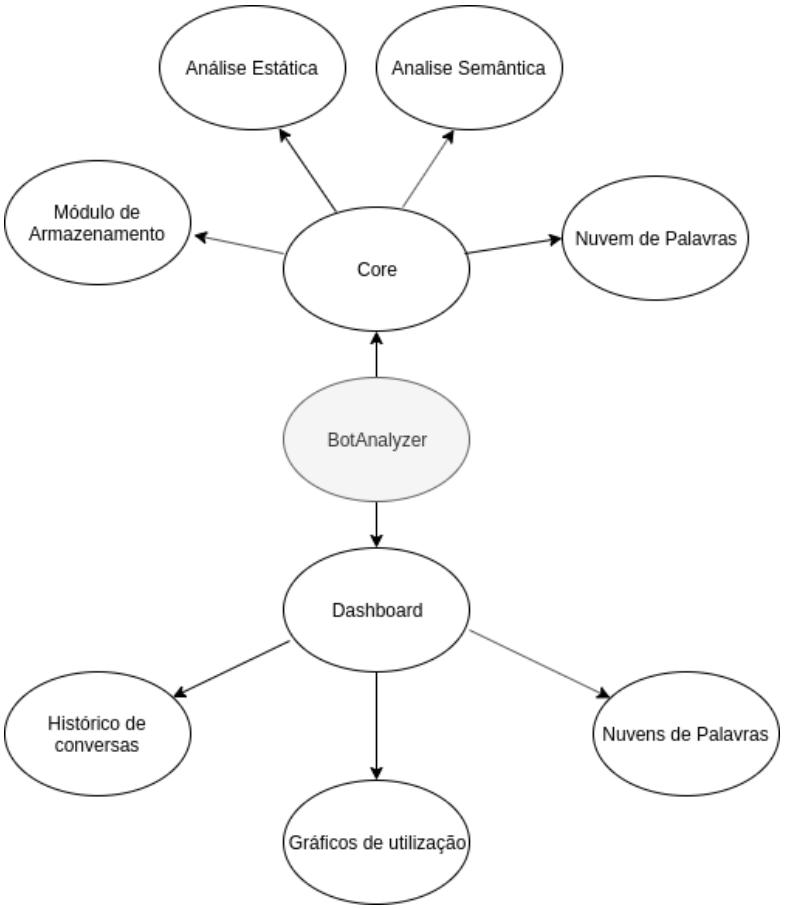
Tabela 1 – Descrição de itens que compõem os históricos de conversação

Item	Descrição
input	Campo responsável por armazenar a pergunta feita pelo aluno
response	Armazena a resposta dada pelo *chatbot*
user_id	Id único para cada aluno que utiliza o *chatbot*
convo_id	Id único para cada sessão aberta da conversa
bot_id	Id do agente conversacional
timestamp	Data e hora da interação no padrão *datetime*

Fonte: Elaborada pelo autor.

Com base na identificação dos itens que compõem o histórico de interação entre um usuário e um *chatbot* foram definidas quais informações seriam interessantes de serem apresentadas ao professor por meio de uma ferramenta interativa. Além disso, foi necessário compreender como apresentar essas informações. Para tal foi necessário definir dois módulos. Um módulo é responsável pelo processamento e armazenamento das informações, denominado *Core*, enquanto o segundo é responsável por organizar e exibir os resultados das análises, denominado *Dashboard*. Uma visão geral sobre os módulos e suas funcionalidades são apresentados na Figura 4.

Figura 4 – Uma visão geral sobre os módulos que constituem a ferramenta BotAnalyzer



Fonte: Elaborada pelo autor.

O módulo Core conta com quatro serviços, são eles:

- ***Módulo de Armazenamento*:** Esse módulo é responsável por mapear os campos contidos no arquivo de *log* e adiciona-los na base de dados da ferramenta, e posteriormente recuperar essas informações para os outros módulos de análise.
- ***Módulo de Análise Estática*:** Esse módulo é responsável gerar as estatísticas de uso do *chatbot*.
- ***Módulo de Análise Semântica*:** Esse módulo é responsável por identificar as perguntas não respondidas nos pares, e então agrupar as perguntas que são semanticamente similares.
- ***Módulo de Nuvem de Palavras*:** Por último, este é responsável por remover as *stop-words* e palavras indesejadas dos textos, contar as palavras semanticamente similares e gerar uma nuvem de palavras com essas informações.

O módulo *Dashboard* contém mecanismos que foram criados afim de mapear as informações do módulo *Core* para os elementos gráficos, e foram divididos com a finalidade de facilitar o reuso e a manutenção do código. São eles:

- ***Histórico de Conversa* :** Esse mecanismo é responsável por organizar as conversar em abas separadas e apresenta-las de forma similar a um *chat*.
- ***Gráficos de Utilização* :** Esse mecanismo é responsável por imprimir os gráficos na tela.
- ***Nuvem de Palavras* :** Esse mecanismo é responsável por criar a nuvem de palavras a partir de uma lista além de criar uma opção para *download* da nuvem de palavras.

Para apoiar a comunicação entre os módulos foi desenvolvida um Api utilizando o modelo de comunicação *Representational State Transfer*(REST)(MASSE, 2011). Este modelo define que cada recurso - funcionalidade do sistema - deve ter uma identificação única para que haja a comunicação e trafego de informações. Dentro desse modelo foram definidas a arquitetura e a modelagem da ferramenta.

3.3 Modelagem da Ferramenta

A primeira característica considerada para ser modelada, foi a forma como os módulos e serviços iriam se comunicar, assim a Figura 5 apresenta a arquitetura proposta e as ações que cada módulo executa.

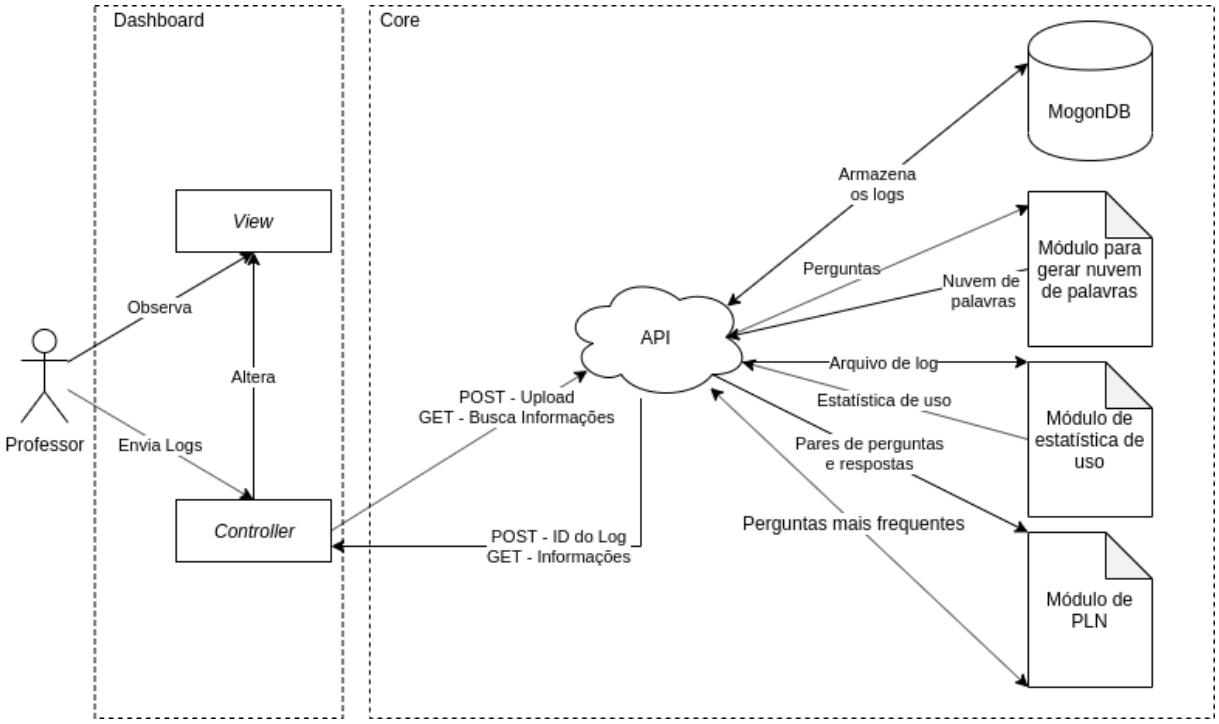
A arquitetura foi desenhada utilizando os padrões do *Service-Oriented Architecture* (SOA), ou Arquitetura Orientada a Serviços (PAPAZOGLOU; HEUVEL, 2007). O SOA baseia-se no conceito do uso de serviços atômicos, independentes e com baixo acoplamento. Como benefícios novas funcionalidades e processos podem se aproveitar das funcionalidades já implementadas apenas sendo necessário construir uma interface nova de comunicação entre o recurso novo e o já existente, e facilita a manutenção e a análise de impacto[1] dessas manutenções, já que os componentes se encontram isolados uns dos outros.

A Figura 5 mostra a arquitetura. Nela o professor atua como ator do sistema interagindo com o *Dashboard*. Por meio do *Dashboard*, o professor pode selecionar o arquivo de *log* que será interpretado. Após a seleção do *log*, o mesmo será enviado para a camada *controller* do módulo *Dashboard*, a camada *controller* envia o arquivo de *logs* através de uma requisição HTTP/POST[2] e com codificação application/x-www-form-urlencoded. Após isso o módulo *Core* salvar o arquivo de *logs*, a requisição é retornada com um identificador para o rastreio do *log*. Quando o *Dashboard* recebe o identificador, ele faz outra requisição para o módulo *Core*, visando recuperar as informações do processamento do log. E seguindo o padrão REST a requisição para recuperar as informações do tipo HTTP/GET

Essa arquitetura de comunicação foi pensada afim de criar uma API que funcione independentemente de uma interface gráfica. Com isso outras pessoas poderia desenvolver suas próprias versões de *dashboard's* apenas utilizando a reposta da API do módulo *Core*.

[1] Consistem em analisar em quais partes da aplicação dependem da parte que será modificada.
[2] Método de requisição projetado para o servidor web aceite os dados anexados no corpo da mensagem

Figura 5 – Arquitetura do BotAnalyzer



Fonte: Elaborada pelo autor.

A Figura 6 apresenta o fluxo de funcionamento do módulo *Core*. De acordo com essa Figura, o fluxo apresenta a caminho que a informação segue após o inicio das duas requisições feitas pelo módulo do *Dashboard*.Nesse fluxo é possível entender as tomadas de decisões que ocorrem durante o processo de recebimento das informações até o momento de retorno destas informações.

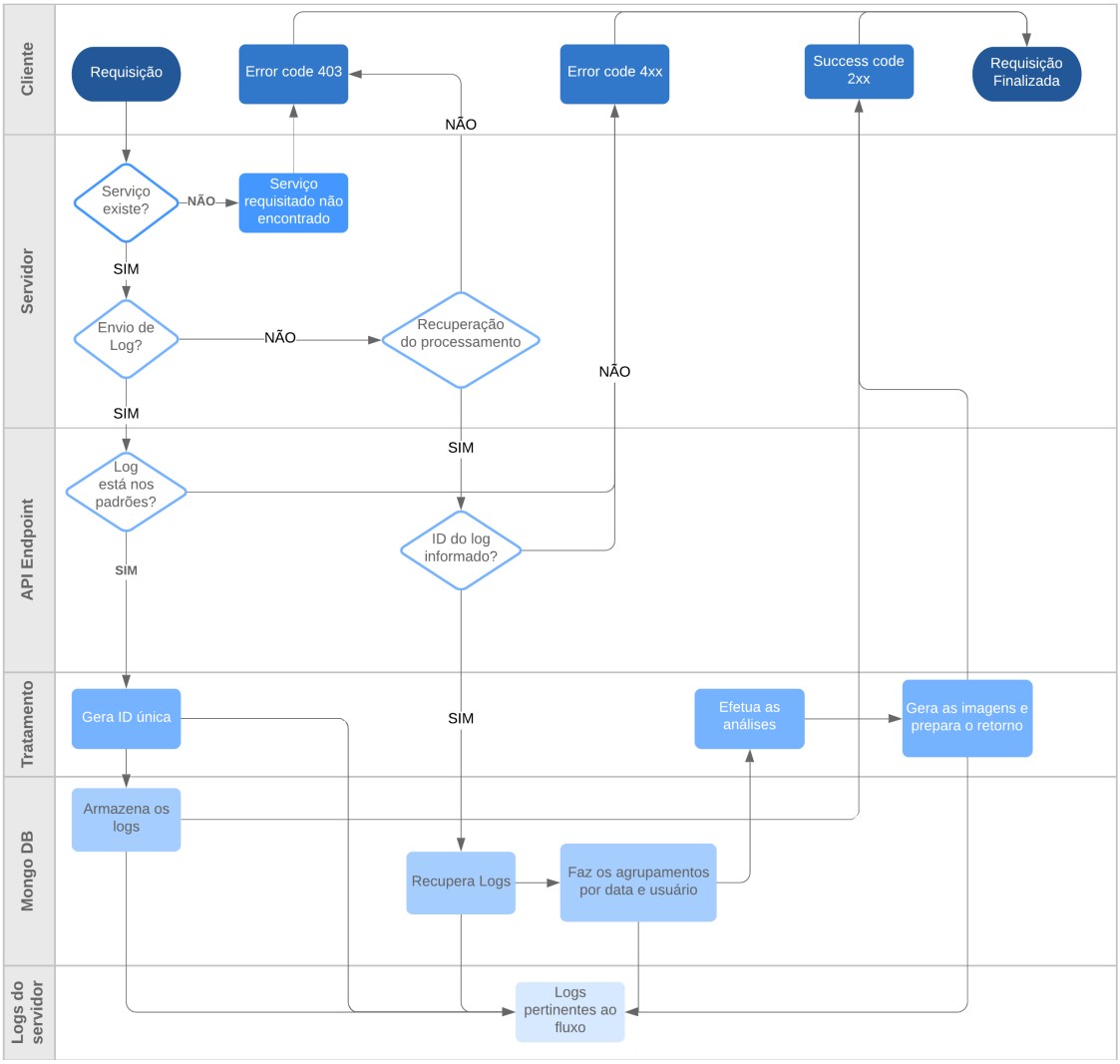
O módulo é dividido em 6 camadas, sendo apenas 5 camadas responsáveis pelo tratamento dos dados e a sexta camada fica responsável apenas pelo gerenciamento dos *logs* gerados pelo módulo. Salienta-se que o módulo foi dividido em camadas afim de entender o fluxo que a informação de entrada deve seguir, porém essas camadas não representam diretamente cada sub-modulo do módulo *Core*. As camadas servidor e *API Endpoint* representam a interface do modulo com o cliente, já os serviços de armazenamento,análise estática, análise semântica e nuvem de palavras se encontram na camada de tratamento.

- ***Camada cliente*:** Esta camada é responsável por fazer a interface entre o Cliente e o modulo, nesta camada ocorrem o gerenciamento das informações dos clientes e o envio do resultado do processamento.

- ***Camada Servidor*:** Esta primeira camada do servidor recebe a requisição, verifica se o serviço solicitado pelo cliente existe na API, verifica se os dados de entrada estão corretor e então passa as informações para a camada *API Endpoint*.

- ***Camada API Endpoint*:** Esta é a ultima camada da API, ela é responsável por chamar os serviços de armazenamento, análise estática, análise semântica e nuvem de palavras.
- ***Camada Tratamento*:** Nesta camada estão contidos os serviços de armazenamento, análise estática, análise semântica e nuvem de palavras.
- ***Camada Banco de Dados*:** Esta camada representa o módulo de armazenamento, ja que nela se encontram o banco de dados e os mecanimos de busca e agrupamento da API
- ***Camada Logs do Servidor*:** Por ultimo a camadas de *Logs* do servidor é uma camada que não fica exposta ao cliente e tem como finalidade armazenar todas as interações que ocorrem no módulo, com a finalidade de manter a rastreabilidade das interações.

Figura 6 – Fluxo de processamento da API



Fonte: Elaborada pelo autor.

Como base de dados foi utilizado um banco do tipo NoSql, para que se seja possível fazer a integração com outros *chatbot* sem que os campos do *logs* ficassem amarrados a uma modelagem rígida, na qual um registro fica relacionado a uma Tabela.

Com a finalidade de descrever o funcionamento do banco de dados, o Código 2 apresenta um exemplo de modelo de registro criado para o *chatbot* TOB-STT

Código-fonte 2: Modelagem do Banco de Dados

```
1 {
2     _id       : <ObjectId>
3     id        : Inteiro,
4     input     : String,
5     response  : String,
6     user_id   : Inteiro,
```

```
7       convo_id : String,
8       bot_id   : Inteiro,
9       timestamp: Data,
10      group    : String,
11  }
```

O campo _id é o identificador criado automaticamente para cada registro adicionado, já o campo *group* armazena o identificador do arquivo de *log* que foi enviado para o servidor. Cada arquivo de *log* recebe um identificador que sera utilizado para recuperar e agrupar todos os registros de conversas salvos no banco de dados. Já os outros campos são descritos nas Tabela 1.

3.4 Etapas de implementação

Esta seção tem como objetivo apresentar as principais etapas que foram necessárias para o desenvolvimento da ferramenta BotAnalyzer. Salienta-se que a implementação foi divida em 4 etapas baseadas nos serviços do módulo *Core*: Armazenamento, Análise Estatística, Análise Semântica e Nuvem de palavras. Além dessas etapas, o desenvolvimento da ferramenta contemplou o estabelecimento de uma interface de usuário (que será descrita na seção 3.5) e com a configuração do ambiente de desenvolvimento (que será descrito na seção 3.6).

3.4.1 Serviço de Armazenamento

O serviço de armazenamento faz uma interface entre o cliente, que no caso dessa aplicação é a pessoal que envia o arquivo de *logs* para o servidor, e a base de dados. Em outras palavras, este serviço é responsável por receber um conjunto de *logs* e então fazer a inserção desse conjunto de informações na base de dados.

A necessidade de utilizar um banco de dados para efetuar o armazenamento dos *logs* das conversas se deu pelo fato de que um banco de dados já possui algumas funcionalidades implementadas, como agrupamento de registros a partir de chaves ou datas e buscas de registros.

Essas funcionalidades de agrupamento, ordenação e contagem de registros providas pelo gerenciados de banco de dados (MongoDB) foram utilizadas para recuperar os registros das conversas e efetuar as operações de agrupamento das conversar e contagem dos registros contidos em cada agrupamento. Essas operações permitiram gerar as informações, tais como contagem de utilização por dia, ou por dia da semana do *chatbot*. Além de realizarem o agrupamento de dados para o processamento estatístico e semânticos dos pares perguntas-respostas.

3.4.2 Serviço de Análise Estatística

Visando entender os momentos em que os alunos mais utilizaram o *chatbot* e visualizar análises estatísticas, tais como, encontrar os termos mais frequentes. Foi necessário criar um

módulo que recuperasse as informações contidas no banco de dados e gerasse as análises pertinentes.

Com o auxilio dos mecanismos já implementados na ferramenta de gerenciamento banco de dados foi possível realizar algumas análises estatísticas para obter informações estatísticas a partir dos logs de conversação do *chatbot*. Elas foram utilizadas para verificar a quantidade de perguntas que não são respondidas pelo *chatbot*, o número de alunos que utilizou o *chatbot*, e gerar os gráficos de utilização do *chatbot* por período.

Para efetuar as análises foram utilizados os comandos de agrupamento e soma do MongoDB, o código 3 oferece um exemplo de uma busca de um *log*, o resultado dessa busca é agrupado nos dias da semana, e com esse resultado o serviço de análise semântica consegue gerar um objeto com uma relação entre os dias da semana e o número de alunos que utilizaram o *chatbot*, o código apresenta um exemplo do objeto gerado pela consulta do código 3

O comando *aggregation*[3], permite uma primeira busca para obter os registros utilizando o identificador do *log* Na sequência, é extraído o dia da semana de cada registro e feito um agrupamento com o dia da semana, utilizando esse novo campo. Com esse comando é possível reconhecer a frequência de uso do *chatbot* pelo usuário por dias da semana.

Código-fonte 3: Exemplo de agregação.

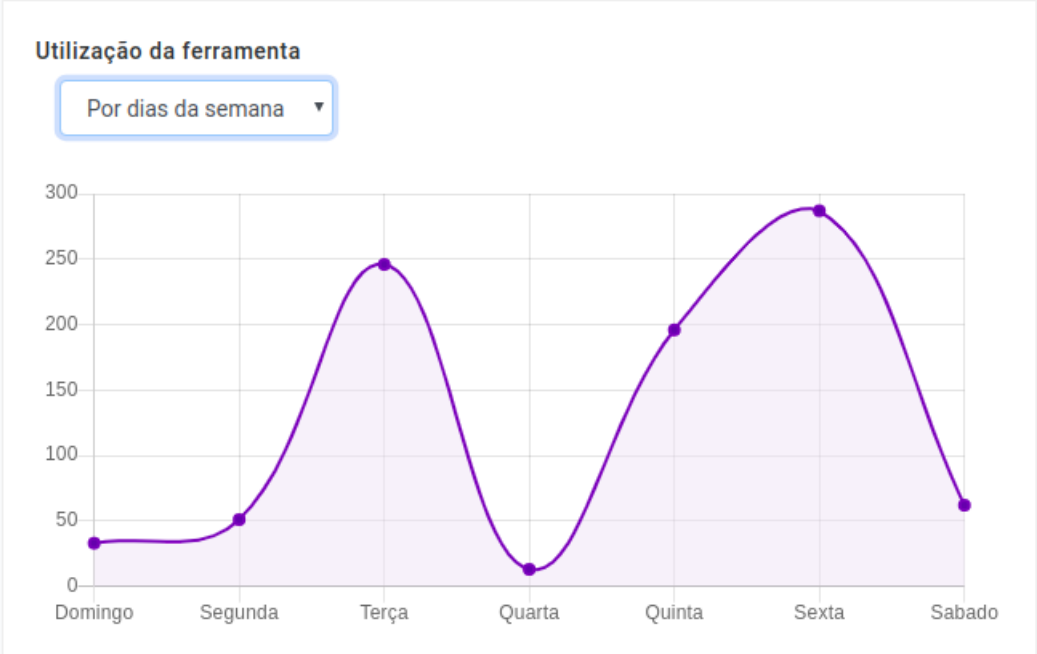
```
1  aggregate([
2          {$match : {
3                  group : uid
4              }
5          },
6          {$project : {
7                  toGroup : {\$dayOfWeek : {$toDate : '$timestamp'}}
8              }
9          },
10         {$group : {
11                 _id : '$toGroup',
12                 count : {$sum : 1}
13             }
14         },
15         {$sort : {
16                 _id : 1
17             }
18         }
19     ]
```

O resultado obtido pelo trecho de código 3 é enviado para o *Dashboard*, que por sua vez gera o Gráfico apresentado na Figura 7, nela é possível observar a frequência de interação

[3] https://docs.mongodb.com/manual/aggregation/

aluno-*chatbot*. O abscisa apresenta o número total de utilização do *chatbot* em um determinado período, enquanto a ordenada apresenta o período, podendo ser dias do mês, dias da semana e horario do dia.

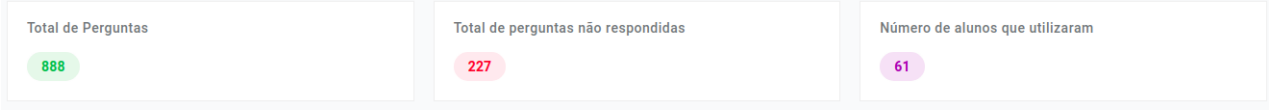
Figura 7 – Gráfico de utilização do *chatbot* por dia da semana



Fonte: Elaborada pelo autor.

A Figura 8 apresenta a quantidade de perguntas que foram feitas pelos alunos, o total de perguntas não respondidas e número de alunos que o utilizaram. Ressalta-se que para os cálculos do número de perguntas que foram feitas pelos alunos e número de alunos que utilizaram o *chatbot* foi necessário utilizar a função de agrupamento $*group* e a função de contagem $*sum* com os campos *_id*, *user_id* mencionados no Código 2.

Figura 8 – Estatísticas de uso do *chatbot*



Fonte: Elaborada pelo autor.

Para obtenção do número de perguntas que não foram respondida pelo *chatbot*, foi necessário utilizar algumas técnicas de comparação semânticas entre textos, para encontrar respostar que se aproximassem semanticamente do padrão *"I don't know how to respond this."*. A próxima seção irá apresentar com mais detalhes a análise.

3.4.2.1 Serviço de Nuvem de palavras

Já no caso da nuvem de palavras, foram utilizados alguns procedimentos diferentes. Primeiro foi necessário quebrar as perguntas feitas pelos alunos em blocos(*tokens*). Para isso, foi utilizado o função *tokenize* da biblioteca NLTK, esta função realiza a separação de uma frase em termos. O segundo passo necessário, está relacionado com a remoção das *stop-words*. Por padrão, palavras *a, but, or...*, são removidas nesse tipo de análise, porém, durante os testes notou-se que alguns termos como *What*, *Could*, permaneceram no conjunto de *tokens*. Para remover estes termos que não têm relevância na geração da núvem de palavras foi necessário adicionar todas as palavras indesejadas e pontuações no conjunto de *stop-words*.

Com o conjunto de termos montados utilizou-se a função *stem*[4] do NLTK para poder encontra as palavras raízes e então agrupa-las. Esta função realiza um processo de *stemming* nas palavras, que consiste em reduzir as palavras "recuperação", "recuperado", "recupera" para o radical "recupera". Nesse sentido, consiste em reduzir os termos encontrados em termos raízes ou base. Com essa operação é possível contabilizar os termos que são igual e assim gerar a nuvem de palavras (Figura 9).

Figura 9 – Nuvem de palavras com os termos mais frequentes.

Fonte: Elaborada pelo autor.

Vale salientar que o tamanho das palavras é proporcional à frequência com que ela aparece, o módulo *Core* envia um vetor que relaciona as palavras com a frequência com que elas aparecem e a interface no módulo *Dashboard* fica responsável por gerar a nuvem]

Visando manter uma boa visibilidade dos termos impressos na nuvem além de trazer uma quantidade suficiente de termos, foram impressas 20 palavras na nuvem (Cui *et al.*, 2010) .

[4] A função *stem* remove os afixos da palavra, deixando apenas a palavra raiz

3.4.3 Serviço de Análise Semântica

O serviço de análise semântica é responsável por receber os pares perguntas e respostas, e interpretar esses pares afim de encontrar as perguntas não respondidas e então agrupar as perguntas não respondidas baseado na similaridade semântica entre elas.

Esse serviço foi codificado utilizando a linguagem de programação Python, e foi utilizando a biblioteca de processamento de linguagem natural NLTK[5] junto da base de dados WordNet[6]. Para a escolha da biblioteca de processamento de linguagem natural foram estudadas algumas bibliotecas: spaCy, NLTK, SyntaxNet, OpenNLP (OMRAN; TREUDE, 2017). Foi observado que a biblioteca NLTK apresenta algumas vantagens nos processos de *tokenização*[7], além de oferecer suporte à WordNet.

Para a implementação da comparação semântica entre textos, foi criado o algoritmo 1. O algoritmo se baseia em três etapas: Na primeira etapa os dois textos a serem comparados são quebrados em *tokens*. Na segunda etapa é encontrada a *synset*[8] de cada *token*. A terceira, e

[5] https://www.nltk.org/
[6] https://wordnet.princeton.edu/
[7] Processo mais utilizado para analise dos textos(OMRAN; TREUDE, 2017)
[8] grupo de elementos de dados que são considerados semanticamente equivalentes

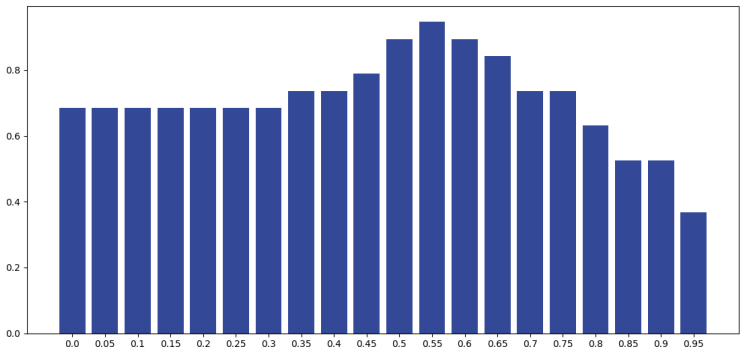
ultima, etapa consiste em encontrar um valor que indica o quão similar dois *tokens* são.

Algoritmo 1: Algoritmo para comparação semântica entre duas frases. sentence_similarity(sentence1,sentence2)

```
   Input: Duas Strings(sentence1,sentence2)
 1 if (sentence1 not String) or (sentence2 not String) then
 2     return
 3 0 new_sentence1 = repeat
 4     s ← tokenize(s)
 5     s ← pos_tag(s)
 6     new_sentence1 += s
 7 until s in sentence1;
 8 new_sentence2 = repeat
 9     s ← tokenize(s)
10     s ← pos_tag(s)
11     new_sentence2 += s
12 until s in sentence2;
13 synsets1 = [] repeat
14     synsets1.append( get_synset(s))
15 until s in new_sentence1;
16 synsets2 = [] repeat
17     synsets1.append( get_synset(s))
18 until s in new_sentence2;
19 sum = 0 count = 0 repeat
20     repeat
21         encontrar o maior caminho de
22         s ← ss
23         sum += maior caminho
24         count += 1
25     until ss in synsets2;
26 until s in synsets1;
27 mean ← sum/count
28 return n1
```

Durante o desenvolvimento do algoritmo de comparação de similaridade entre textos 1 foi necessário efetuar algumas simulações para encontrar qual porcentagem de similaridade gerada pelo algoritmo traria a maior taxa de acerto. Para isso foi criado uma base de teste contendo duas frases e qual seria a resposta esperada. Durante a simulação, a porcentagem de similaridade, que representa o ponto de decisão para dizer se duas frases são similares ou não são similares, foi variada de 0 a 100% com um passo de 0.5%, obtendo o gráfico da Figura 10.

Figura 10 – Gráfico com a taxa de acerto em cada ponto de decisão



Fonte: Elaborada pelo autor.

Conforme é possível observar na Figura 10, a porcentagem que trouxe o maior numero de acertos é de 55%. Nesse sentido, todos os pares de frases com um grau de similaridade de pelo menos 55% são semelhantes.

A Figura 11 apresenta um exemplo de uma tabela gerada pelo serviço. Nela é possível visualizar as perguntas não respondidas e número de vezes q elas aconteceram.

Figura 11 – Tabela de top 10 perguntas não respondidas

Perguntas mais frequentes(top 10).

Pergunta mais frequente não respondida — 8 x

What is an equivalence class?

Pergunta	Status	Numero de vezes
What is an equivalence class?	Não Respondida	8 x
Show me an instance of boundary value analysis	Não Respondida	6 x
Whats is a class?	Não Respondida	6 x
What a error?	Não Respondida	5 x
Tell me this	Não Respondida	5 x
Tell me	Não Respondida	4 x
Software testing	Não Respondida	3 x
Show me a instance of functional testing	Não Respondida	3 x
Instance of boundary values analysis	Não Respondida	1 x

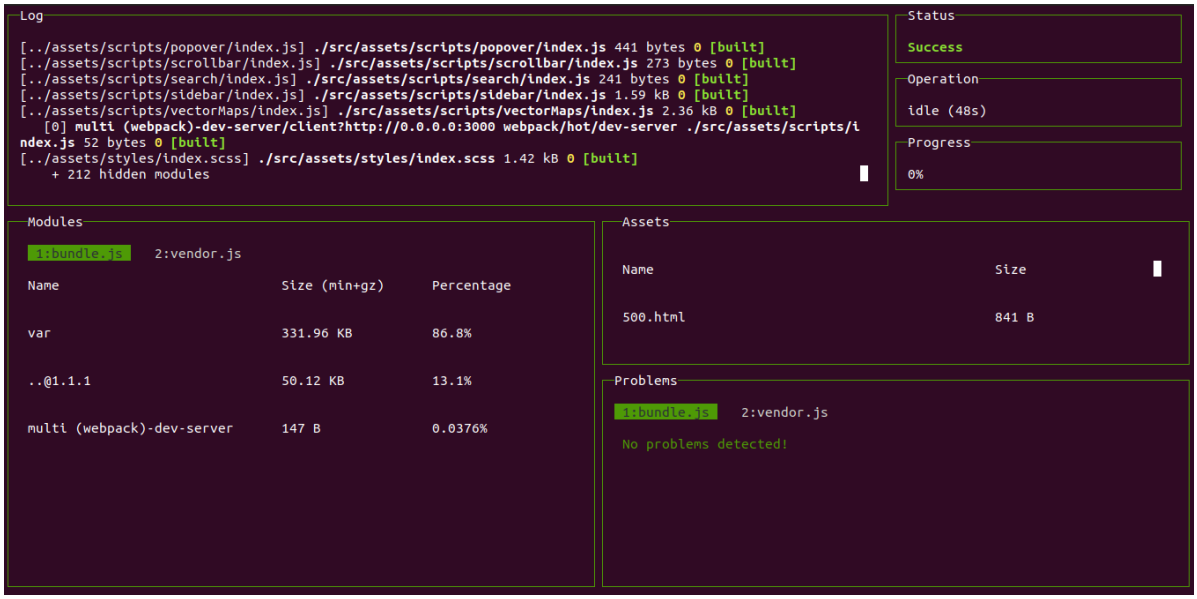
Ver todas as perguntas

Fonte: Elaborada pelo autor.

3.5 Criação da Interface

Para apoiar o desenvolvimento da interface de usuário, foram utilizados dois *frameworks*. O primeiro é o gerenciador de pacotes JavaScript NPM[9] e o segundo é o WebPack[10]. O NPM permite a instalação de componentes e elementos gráficos, tais como modelos de gráficos, campos de *upload* de arquivos, etc. O WebPack, por outro lado, permite juntar os pacotes Javascript produzidos pelo baixados pela ferramenta NLTK e os códigos escritos pelo altos. Além disso, a WebPack fornece *dashboards* com as informações de utilização, tais como memória utilizada e tempo de execução, além de facilitarem a visualização de *logs* de erro. Na Figura 12 é apresentado um exemplo do *dashboards* do *framework* WebPack.

Figura 12 – Painel de auxilio ao desenvolvimento da ferramenta Webpack



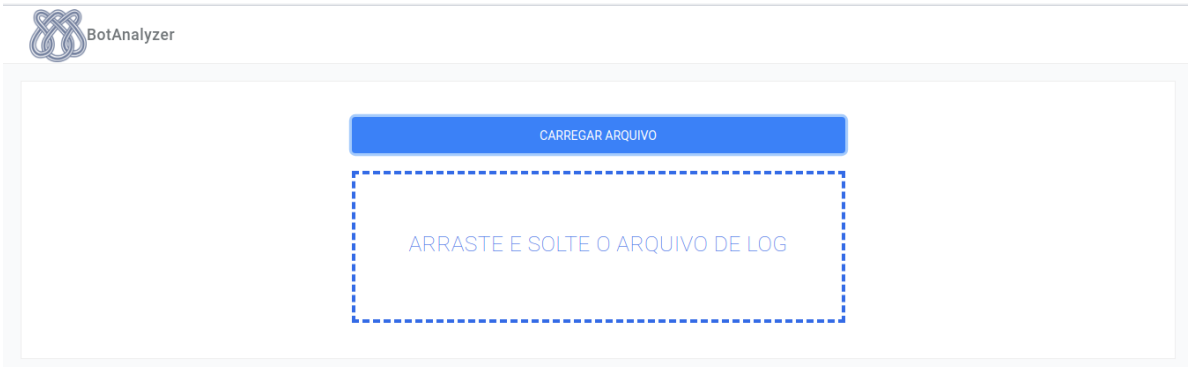
Fonte: Elaborada pelo autor.

Para a criação da UX/UI foi utilizado um *template* de painel de administração chamado Adminator[11], este *template* oferece as configurações de estilo do site através de uma folha de estilos(css)[12] além de utilizar as ferramentas NPM e WebPack.

A Figura 13, por exemplo, apresenta a interface para *upload* do *log*, nela é possível carregar o arquivo apenas arrastando-o para dentro do quadrado pontilhando, ou selecionando o arquivo através do botão "Carregar Arquivo".

9 https://www.npmjs.com/
10 https://webpack.js.org/
11 https://github.com/puikinsh/Adminator-admin-dashboard
12 DescriçãoCascading Style Sheets é um mecanismo para adicionar estilo a um documento web.

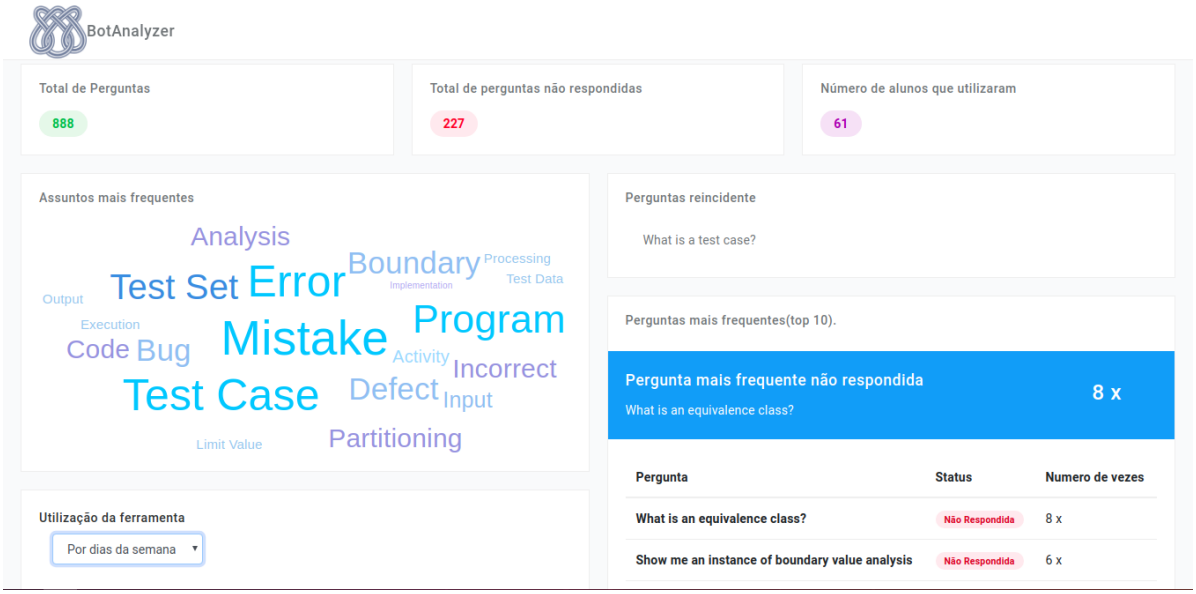
Figura 13 – Tela de *upload* do arquivo



Fonte: Elaborada pelo autor.

Após o envio do arquivo, o *dashboard* já esta configurado para efetuar as etapas de comunicação com o módulo *Core*, citadas na seção 3.3, sendo assim, ele exibe a tela com as informações da interpretação feita pelo módulo *Core*, como exemplificado na Figura 14.

Figura 14 – Tela principal do *Dashboard*



Fonte: Elaborada pelo autor.

3.6 Configuração da Ferramenta BotAnalyzer

A configuração da ferramenta BotAnalyzer baseia-se em duas etapas, a primeira consiste em configurar o módulo *Core* e suas dependências, e depois gerar uma *build* do dashboard que será utilizado. Após os passos anteriores é necessário conFigurar as portas de comunicação para que o módulo *Core* possa se comunicar via *Http Request* com o *Dashboard*.

O módulo *Core* foi desenvolvido utilizando o *framework* ExpressJS[13]. Esse *framework* possibilita a criação de uma aplicação na plataforma Node.JS[14]. Essa plataforma utiliza a linguagem de programação JavaScript para a criação de suas aplicações.

A vantagem de sua utilização se encontra em sua velocidade de execução das tarefas, já que ela executa de forma assíncrona todas as tarefas. Além disso a plataforma possui uma arquitetura voltada a eventos, onde um processo pai dispara a chamada para os processos filhos - neste caso são os serviços - e espera o retorno deles, facilitando a implementação da arquitetura proposta.

Além disso, o módulo *core* depende da instalação da base de dados NoSql[15] MongoDB para o seu funcionamento, após a instalação e configuração é necessário adicionar as credenciais e rotas do banco de dados no arquivo de configuração do módulo.

Para manter o módulo *Core* funcionando é necessário utilizar uma ferramenta *comand-line interface* (CLI) para manter o módulo funcionando continuamente e gerenciar os *logs* de saída do módulo. Foi utilizado a ferramenta Forever[16], a escolha da ferramenta de seu pela compatibilidade dela com a plataforma Node.JS. A configuração consiste em um arquivo JSON com as informações de caminho para o servidor, caminhos para os arquivos de *log* e nome da aplicação, como exemplificado no código 5.

Código-fonte 4: Exemplo de arquivo de configuração

```
1 {
2     "uid": "app", //Nome da aplicação
3     "append": true, //Indica se os logs serão adicionados ao final do
      arquivo ou se será criado um novo arquivo
4     "script": "index.js", //Nome do arquivo Node.JS
5     "sourceDir": "/home/myuser/app", //Caminho absoluto para o
     arquivo de script
6     //Os três campos indicam os caminhos para os arquivos de log
7     "logFile": "/home/myuser/logs/forever.log",
8     "outFile": "/home/myuser/logs/out.log",
9     "errFile": "/home/myuser/logs/error.log"
10 }
```

Já o módulo de *Dashboard* utiliza um servidor Httpd Apache, ele é responsável por disponibilizar as páginas e todos os recursos que o professor pode acessar no *Dashboard*, como a interface do *Dashboard* é compilada, gerando um site, como descrito na sub seção 3.5, não é

[13] https://nodejs.org/en/

[14] É um interpretador assíncrono que utiliza a linguagem de programação JavaScript. Esse interpretador utiliza o paradigma de programação orientada a eventos.

[15] *Not Only Sql*, são banco de dados não relacional, ou seja, fornecem um mecanismo de armazenamento e recuperação de dados que não são tabulares

[16] https://www.npmjs.com/package/forever

necessário efetuar nenhuma configuração no módulo apenas as configurações de instalação do Servidor Apache.

3.7 Exemplo de Uso

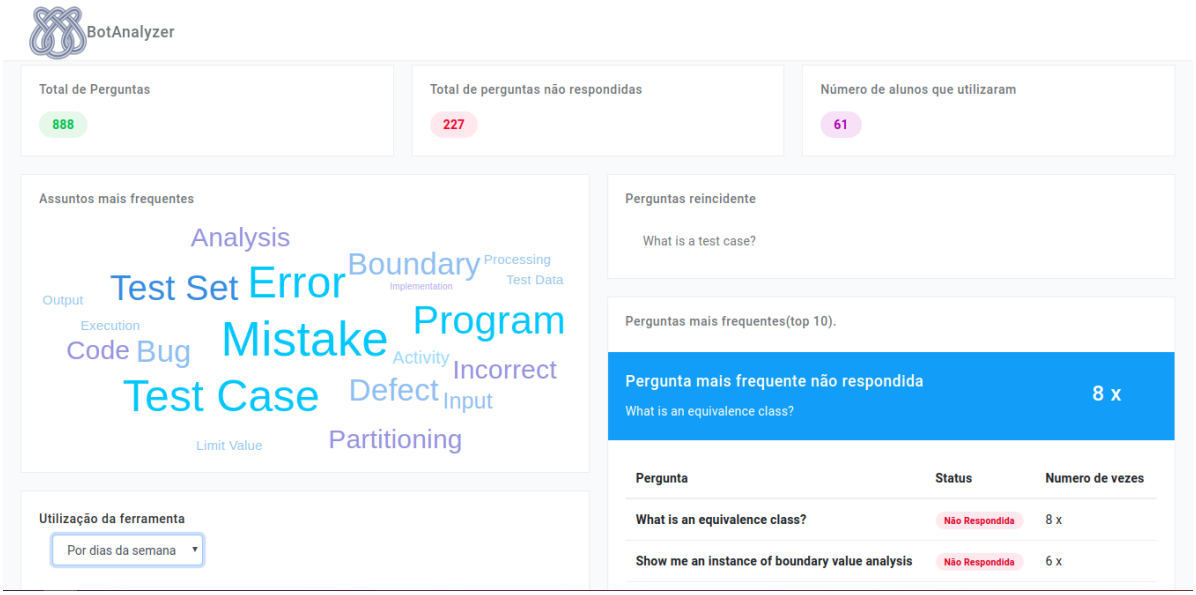
A fim de avaliar o uso do BotAnalyzer, foi conduzida uma avaliação na qual os alunos utilizaram o *chatbot* TOB-STT para solucionar suas dúvidas sobre teste e inspeção de *software*. A avaliação foi realizada na disciplina SSC0721 - Teste e Inspeção de Software, oferecida aos alunos de Engenharia de Computação e Ciências da Computação. A disciplina contava com 36 alunos de graduação e 14 alunos de pós graduação e foi ministrada pela professora Simone Senger de Souza.

Inicialmente, foi preparado um documento (Apêndice A) sobre teste e inspeção de software e o alunos deveriam se posicionar como profissionais de *Quality Assurance*(QA)[17] e inspecionar esse documento, buscando defeitos conceituais relacionados a teste de software e classificando-os nos tipos de defeitos definidos para inspeção de *software*. Para a realização do experimento, o aluno de iniciação científica da professora Simone Senger de Souza, Lucas Turci, apresentou o funcionamento do *chatbot* TOB-STT aos alunos da disciplina SSC0721. Após essa apresentação os alunos tinham como tarefa inspecionar o documento fornecido, podendo utilizar o *TOB-STT* para tirar suas dúvidas sobre os conceitos de teste de software.

Os dados do uso do TOB-STT foram coletados e analisados por meio do BotAnalyzer. Durante o uso, notou-se que alguns alunos tentaram "testar"o TOB-STT, fazendo perguntas fora do escopo de teste de software. Isso acabou gerando várias perguntas não respondidas pelo *chatbot*. Ainda assim foi possível analisar o comportando do *chatbot* e entender quais foram as dúvidas que geraram as maiores solicitações ao agente conversacional. A Figura 15, apresenta a primeira parte do *dashboard*, onde é possível visualizar a nuvem de palavras com os tópicos mais perguntados e as perguntas não respondidas.

[17] É um conjunto de atividades que visa garantir que um produto ou serviço esteja de acordo com o nível de qualidade exigido

Figura 15 – Resultado da análise dos *logs* do experimento



Fonte: Elaborada pelo autor.

Após o experimento, foi solicitado que os alunos respondessem a um questionário para entender fosse possível analisar a percepção deles em relação ao *chatbot*. Utilizando a correção das respostadas do exercício aplicado, foi possível perceber que a maioria dos alunos conseguiu identificar entre 40% e 50% dos defeitos que foram injetados(PASCHOAL *et al.*, 2019). Além da correção do exercício, foram analisadas as respostas dos alunos ao questionário, afim de compreender qual foi a percepção dos alunos sobre o uso do agente(PASCHOAL *et al.*, 2019).

Comparando os resultados obtidos na Figura 15 e no experimento de Paschoal *et al.* (2019), pudemos identificar que o mecanismo para interpretar os *logs* conseguiu, de uma forma geral, interpretar as dúvidas e os problemas que os alunos tiveram ao utilizar o agente conversacional TOB-STT. Dessa forma o professor pode entender de uma forma simples quais são os problemas que os alunos tiveram, diminuindo a lacuna entre o professor e o alunos gerada pela utilização do *chatbot*.

Este exemplo de uso é simples, mas é possível visualizar como o BotAnalyzer pode ser útil ao professor, trazendo-lhes informação sobre as dúvidas dos alunos enquanto estudam para a disciplina. Com os recursos oferecidos pelo BotAnalyzer é possível que o professor:

- Visualize, pela nuvem de palavras, os itens que geraram mais dúvidas e assim guiar o professor para que possa preparar material ou aula para esclarecer dúvidas pendentes;
- Analise perguntas não respondidas, tentando caracterizar se são relevantes para o contexto da disciplina, podendo usar essa informação para reforçar esses assuntos em sala de aula;
- Visualize a quantidade de interações feitas pelos alunos, por meio da quantidade de perguntas feitas ao *chatbot*;

- Ter informação estatística sobre em que momento do período da atividade os alunos mais interagiram com o *chatbot*.

Essas e outras informações são úteis para direcionar o professor, principalmente em cenários de metodologias ativas como Flipped Classroom. Ainda faltam estudos sobre a percepção do professor com o uso do BotAnalyzer, o que está sendo planejado para as próximas atividades relacionados à evolução dessa ferramenta.

3.8 Considerações Finais

A ferramenta visa aliviar a carga de trabalho do professor, ao passo que ela traz de uma forma fácil de visualização de todas as duvidas e problemas que os alunos tiveram durante os estudos. Com essas informação, os professores podem se preparar previamente para as aulas, sem que seja necessário ler um extenso arquivo de *log*.

Capítulo 4

CONCLUSÃO

4.1 Contribuições

Este trabalho apresenta o estudo e desenvolvimento de uma ferramenta de apoio ao processamento de conversas entre aluno-*chatterbot*, denominada BotAnalyzer, a qual apresenta uma interface por meio de dashboard com informações sobre o uso de *chatterbot* pelos estudantes. O desenvolvimento deste trabalho foi bastante desafiador pois envolveu conceitos de processamento de linguagem natural e conhecimento sobre estruturas de *chatterbot*.

O BotAnalyzer foi avaliado no contexto de uma disciplina de teste de software e os resultados indicaram que o mesmo pode auxiliar positivamente o docente com os resultados da interação dos estudantes, por exemplo, vendo as maiores dificuldades dos mesmos, as quais aparecem devido às interações realizadas. Como trabalhos futuros, pretende-se avaliar o BotAnalyzer no contexto de um curso que utilize *FlippedClassroom* obtendo-se, desta forma, um maior número de interações e observando como o BotAnalyzer pode colaborar com essa metodologia de ensino.

REFERÊNCIAS

AGASSI, J.; WIEZENBAUM, J. Computer power and human reason: From judgment to calculation. **Technology and Culture**, v. 17, p. 813, 10 1976. Citado na página 25.

ALENCAR, M. A. dos S.; NETTO, J. F. de M. Cyberpoty: Um chatterbot 3d para interação com usuários de um portal de educação a distância. In: **Anais do Workshop de Informática na Escola**. [S.l.: s.n.], 2010. v. 1, n. 1, p. 1417–1420. Citado na página 23.

ANDREEVSKAIA, A.; BERGLER, S. Clac and clac-nb: Knowledge-based and corpus-based approaches to sentiment tagging. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 4th international workshop on semantic evaluations**. [S.l.], 2007. p. 117–120. Citado na página 31.

BENAHMED, Y.; SELOUANI, S. A.; O'SHAUGHNESSY, D. D. Ontology-based pattern generator and root semantic analyser for spoken dialogue systems. 04 2012. Citado na página 63.

BOLLWEG, L.; KURZKE, M.; SHAHRIAR, K. A.; WEBER, P. When robots talk-improving the scalability of practical assignments in moocs using chatbots. In: ASSOCIATION FOR THE ADVANCEMENT OF COMPUTING IN EDUCATION (AACE). **EdMedia+ Innovate Learning**. [S.l.], 2018. p. 1455–1464. Citado na página 23.

BRAUN, A. **Chatbots in der Kundenkommunikation**. [S.l.]: Springer-Verlag, 2013. Citado na página 23.

CHEN, L.; SYCARA, K. Webmate: A personal agent for browsing and searching. In: **Agents**. [S.l.: s.n.], 1998. v. 98, p. 132–139. Citado na página 32.

CHOMSKY, N. A note on phrase structure grammars. **Information and control**, Elsevier, v. 2, n. 4, p. 393–395, 1959. Citado na página 27.

CLARK, C. F. A.; LAPPIN, S. **The Handbook of Computational Linguistics and Natural Language Processing**. [S.l.]: Wiley, 2015. Citado 2 vezes nas páginas 27 e 29.

COVINGTON, M. A.; NUTE, D.; VELLINO, A. **Prolog programming in depth**. [S.l.]: Prentice-Hall, Inc., 1996. Citado na página 26.

Cui, W.; Wu, Y.; Liu, S.; Wei, F.; Zhou, M. X.; Qu, H. Context preserving dynamic word cloud visualization. In: **2010 IEEE Pacific Visualization Symposium (PacificVis)**. [S.l.: s.n.], 2010. p. 121–128. Citado na página 46.

DALY, C.; GIBSON, W. P.; TAYLOR, G. H.; JOHNSON, G. L.; PASTERIS, P. A knowledge-based approach to the statistical mapping of climate. **Climate research**, v. 22, n. 2, p. 99–113, 2002. Citado na página 31.

GALVAO, A. M.; BARROS, F. A.; NEVES, A. M.; RAMALHO, G. L. Persona-aiml: an architecture for developing chatterbots with personality. In: IEEE. **Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.** [S.l.], 2004. p. 1266–1267. Citado na página 25.

GROUP, X. R. *et al.* A lexicalized tree adjoining grammar for english. **arXiv preprint cs/9809024**, 1998. Citado na página 30.

HAKEN, H. **Synergetic computers and cognition: A top-down approach to neural nets**. [S.l.]: Springer Science & Business Media, 2013. v. 50. Citado na página 31.

HARE, J. S.; SINCLAIR, P. A.; LEWIS, P. H.; MARTINEZ, K.; ENSER, P. G.; SANDOM, C. J. Bridging the semantic gap in multimedia information retrieval: Top-down and bottom-up approaches. 2006. Citado na página 31.

HARRISON, M. A. **Introduction to Formal Language Theory**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1978. ISBN 0201029553. Citado na página 27.

HOPCROFT RAJEEV MOTWANI, J. D. U. J. E. **Introduction to Automata Theory, Laguages, and Computation**. [S.l.]: Pearson, 2001. Citado na página 29.

JONES, K. S. A statistical interpretation of term specificity and its application in retrieval. **Journal of documentation**, Emerald Group Publishing Limited, 2004. Citado na página 32.

JOSHI, A. K. An introduction to tree adjoining grammars. **Mathematics of language**, John Benjamins, Amsterdam, v. 1, p. 87–115, 1987. Citado na página 29.

JOSHI, A. K.; SCHABES, Y. Tree-adjoining grammars. In: **Handbook of formal languages**. [S.l.]: Springer, 1997. p. 69–123. Citado na página 29.

JURAFSKY, J. H. M. D. **Speech and Language Processing**: An introduction to natural language processing, computational linguistics, and speech recognition. [S.l.: s.n.], 2018. Citado na página 26.

KRASSMANN, A. L.; HERPICH, F.; SILVA, Á. S. P. da; SILVA, A. R. da; ABREU, C. de S.; SCHMITT, M. A. R.; BERCHT, M.; TAROUCO, L. M. R. Fastaiml: uma ferramenta para apoiar a geração de base de conhecimento para chatbots educacionais. **RENOTE**, v. 15, n. 2, 2017. Citado 2 vezes nas páginas 24 e 36.

KRASSMANN, A. L.; PAZ, F. J.; SILVEIRA, C.; TAROUCO, L. M. R.; BERCHT, M. Conversational agents in distance education: Comparing mood states with students' perception. **Creative Education**, Scientific Research Publishing, v. 9, n. 11, p. 1726, 2018. Citado na página 35.

LEACOCK, C.; CHODOROW, M. Combining local context and wordnet similarity for word sense identification. **WordNet: An electronic lexical database**, v. 49, n. 2, p. 265–283, 1998. Citado na página 32.

LEMOS, C. E. **Desenvolvimento de *Chatterbot* Educacionais: Um Estudo de Caso Voltado ao Ensino de Algoritmos**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, Natal - RN,, 2011. Citado na página 26.

LEONHARDT, M. D. **Doroty : um chatterbot para treinamento de profissionais atuantes no gerenciamento de redes de computadores.** Dissertação (Mestrado) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre - RS,, 2005. Citado na página 26.

LEONHARDT, M. D.; CASTRO, D. D. d.; DUTRA, R. L. d. S.; TAROUCO, L. M. R. Elektra: Um chatterbot para uso em ambiente educacional. **RENOTE: revista novas tecnologias na educação [recurso eletrônico]. Porto Alegre, RS**, 2003. Citado 2 vezes nas páginas 23 e 35.

LIN, L.; ATKINSON, R. K.; CHRISTOPHERSON, R. M.; JOSEPH, S. S.; HARRISON, C. J. Animated agents and learning: Does the type of verbal feedback they provide matter? **Computers & Education**, v. 67, p. 239 – 249, 2013. Citado na página 26.

MASSARO, D. W.; LIU, Y.; CHEN, T. H.; PERFETTI, C. A multilingual embodied conversational agent for tutoring speech and language learning. In: **Ninth International Conference on Spoken Language Processing**. [S.l.: s.n.], 2006. Citado na página 23.

MASSE, M. **REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces**. [S.l.]: "O'Reilly Media, Inc.", 2011. Citado na página 39.

MAULDIN, M. L. Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In: **Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)**. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994. (AAAI '94), p. 16–21. ISBN 0-262-61102-3. Disponível em: <http://dl.acm.org/citation.cfm?id=199288.199285>. Citado na página 23.

MIHALCEA, R.; CORLEY, C.; STRAPPARAVA, C. *et al.* Corpus-based and knowledge-based measures of text semantic similarity. In: **Aaai**. [S.l.: s.n.], 2006. v. 6, n. 2006, p. 775–780. Citado 2 vezes nas páginas 32 e 33.

MILLER, G. A. Wordnet: a lexical database for english. **Communications of the ACM**, ACM, v. 38, n. 11, p. 39–41, 1995. Citado 2 vezes nas páginas 31 e 32.

MORATO, E. M.; KOCH, I. V. *et al.* Linguagem e cognição: os (des) encontros entre a lingüística e as ciências cognitivas. **Cadernos de Estudos Lingüísticos**, 2011. Citado na página 27.

NEVES, A. M. M.; BARROS, F. A.; HODGES, C. iaiml: a mechanism to treat intentionality in aiml chatterbots. In: **2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)**. [S.l.: s.n.], 2006. p. 225–231. ISSN 1082-3409. Citado na página 25.

OMRAN, F. N. A. A.; TREUDE, C. Choosing an nlp library for analyzing software documentation: a systematic literature review and a series of experiments. In: IEEE PRESS. **Proceedings of the 14th International Conference on Mining Software Repositories**. [S.l.], 2017. p. 187–197. Citado na página 47.

PAPAZOGLOU, M. P.; HEUVEL, W.-J. V. D. Service oriented architectures: approaches, technologies and research issues. **The VLDB journal**, Springer, v. 16, n. 3, p. 389–415, 2007. Citado na página 39.

PASCHOAL, L. N. **Contribuições ao ensino de teste de software com o modelo flipped classroom e um agente conversacional**. Tese (Doutorado) — Universidade de São Paulo, 2019. Citado 2 vezes nas páginas 23 e 26.

PASCHOAL, L. N.; CHICON, P.; KRASSMANN, A.; BINELO, M. Ubibot: Agente inteligente consciente do contexto de aprendizagem do usuário integrado ao ambiente moodle. In: **Anais do XXI Congreso Internacional de Informática Educativa, TISE**. [S.l.: s.n.], 2016. Citado 3 vezes nas páginas 23, 26 e 35.

PASCHOAL, L. N.; CHICON, P. M. M.; FALKEMBACH, G. A. M. Concepção, implementação e avaliação de um agente conversacional com suporte à aprendizagem ubíqua. **RENOTE**, v. 15, n. 1, 2017. Citado na página 23.

PASCHOAL, L. N.; SOUZA, S. d. R. S. d. Como ensinar teste de software com flipped classroom? **Anais**, 2018. Citado na página 23.

PASCHOAL, L. N.; TURCI, L. F.; CONTE, T. U.; SOUZA, S. R. S. Towards a conversational agent to support the software testing education. In: **Proceedings of the XXXIII Brazilian Symposium on Software Engineering**. New York, NY, USA: ACM, 2019. (SBES 2019), p. 57–66. ISBN 978-1-4503-7651-8. Disponível em: <http://doi.acm.org/10.1145/3350768.3352456>. Citado 3 vezes nas páginas 26, 35 e 54.

PAZ, F. J.; SILVEIRA, C.; KRASSMANN, A.; TAROUCO, L. M. R. Perspectivas tecnológicas para o aprimoramento de chatbots educacionais em aiml. **TE & ET**, 2017. Citado na página 36.

RADZIWILL, N. M.; BENTON, M. C. Evaluating quality of chatbots and intelligent conversational agents. **CoRR**, abs/1704.04579, 2017. Disponível em: <http://arxiv.org/abs/1704.04579>. Citado na página 26.

RESNIK, P. Using information content to evaluate semantic similarity in a taxonomy. **arXiv preprint cmp-lg/9511007**, 1995. Citado na página 31.

RICHARDSON, M.; BURGES, C. J.; RENSHAW, E. Mctest: A challenge dataset for the open-domain machine comprehension of text. In: **Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing**. [S.l.: s.n.], 2013. p. 193–203. Citado na página 31.

SANTOS, L. M. A. A inserção de um agente conversacional animado em um ambiente virtual de aprendizagem a partir da teoria da carga cognitiva. 2009. Citado na página 23.

SHAWAR, B.; ATWELL, E. Chatbots: Are they really useful? **LDV Forum**, v. 22, p. 29–49, 01 2007. Citado na página 23.

SUBRAMANIAN, A.; TAMAYO, P.; MOOTHA, V. K.; MUKHERJEE, S.; EBERT, B. L.; GILLETTE, M. A.; PAULOVICH, A.; POMEROY, S. L.; GOLUB, T. R.; LANDER, E. S. *et al.* Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 102, n. 43, p. 15545–15550, 2005. Citado na página 31.

TURNEY, P. D. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In: SPRINGER. **European conference on machine learning**. [S.l.], 2001. p. 491–502. Citado na página 31.

WEIZENBAUM, J. Eliza—a computer program for the study of natural language communication between man and machine. **Commun. ACM**, ACM, New York, NY, USA, v. 9, n. 1, p. 36–45, jan. 1966. Citado 2 vezes nas páginas 23 e 25.

GLOSSÁRIO

Agentes de usuário: qualquer *software* que recupera, processa e facilita a interação do usuário final com o conteúdo Web. Como exemplo desses agentes, podem ser citados navegadores, reprodutores multimídia e tecnologias assistivas.

AIML: é uma linguagem de marcação de texto derivada do XML,considerado por Benahmed, Selouani e O'Shaughnessy (2012) como um *framework* . Ela é constituida por um conjunto de *tags*, que é utilizado para implementar a base de conheciento do *chatterbot*.

Aprendizado de Máquina: .

Corpora: Representa o plural em latim do termo *corpus*.

Corpora: plural do termo *corpus* - Conjunto de dados linguísticos pertencentes ao uso oral ou escrito da língua e que podem ser processados por computado.

Corpus: É um conjunto grande e estruturado de textos.

Framework: é uma abstração que une códigos comuns entre vários projetos de *software* provendo uma funcionalidade genérica. *Frameworks* são projetados com a intenção de facilitar o desenvolvimento de *software*, habilitando designers e programadores a gastarem mais tempo determinando as exigências do *software* do que com detalhes de baixo nível do sistema.

Javascript: é uma linguagem de *script* para desenvolvimento de certos tratamentos que ocorrem lado do cliente, geralmente o navegador Web. Ela é utilizada geralmente quando é inconveniente ou impossível para o servidor para fazer esse tratamento.

NoSQL: *Not Only Sql*, são bancos de dados não relacional, ou seja, fornecem um mecanismo de armazenamento e recuperação de dados que não são tabulares.

Padrões de projeto: ou *Design Pattern*, descreve uma solução geral reutilizável para um problema recorrente no desenvolvimento de sistemas de *software* orientados a objetos. Não é um código final, é uma descrição ou modelo de como resolver o problema do qual trata, que pode ser usada em muitas situações diferentes.

Template: é um documento sem conteúdo, com apenas a apresentação visual (apenas cabeçalhos por exemplo) e instruções sobre onde e qual tipo de conteúdo deve entrar a cada parcela da apresentação.

Web: Sinônimo mais conhecido de *World Wide Web* (WWW). É a interface gráfica da Internet que torna os serviços disponíveis totalmente transparentes para o usuário e ainda possibilita a manipulação multimídia da informação.

APÊNDICE A

EXERCÍCIO DE INSPEÇÃO

A.1 Exercício

Vocês são responsáveis pela equipe de QA de uma empresa de desenvolvimento de software. Vocês precisam convencer a gerência da empresa de que teste de software deve ser priorizado e para isso, solicitaram que um dos testadores organizassem um documento para explicar como teste de software funciona. O papel de vocês agora é inspecionar este documento, na perspectiva de testador, procurando possíveis defeitos no documento. Para isso, utilize o seu conhecimento sobre teste de software e, como apoio, o TOB-SST.

Os defeitos encontrados devem ser inseridos na tabela a seguir. Quando encontrar um defeito no texto, sublinhar o texto e colocar um número para associa-lo a tabela fornecida (na coluna “Identificação do Defeito”). Em “Descrição”, explicar porque você considera isso como um defeito e qual seria o correto. *O número de linhas da tabela não significa o total de defeitos*.

A.2 Documento a ser revisado

Software testing consists of the dynamic verification that a program provides expected behaviors on an infinite set of test cases, suitably selected from the usually finite execution domain.

However, one of the most important limitations of software testing is that testing can show only the presence of failures, not their absence. This is a fundamental, theoretical limitation; generally speaking, the problem of finding all failures in a program is undecidable. Testers often call a successful (or effective) test one that finds an error. In software testing, the following terms are employed:

- **Software Fault:** A static defect (or bug) in the software.
- **Software Error:** An incorrect internal state that is the manifestation of some fault
- **Software Failure:** External, incorrect behavior with respect to the requirements or other description of the expected behavior.

To illustrate, consider a medical doctor making a diagnosis for a patient. The patient enters the doctor's office with a list of failures (that is, symptoms). The doctor then must discover the fault, or root cause of the symptom. To aid in the diagnosis, a doctor may order tests that look for anomalous internal conditions, such as high blood pressure, an irregular heartbeat, high levels of blood glucose, or high cholesterol. In our terminology, these anomalous internal conditions correspond to errors. While this analogy may help us thinking about faults, errors, and failures, software testing and a doctor's diagnosis differ in one crucial way. Specifically, faults in software are design mistakes. They do not appear spontaneously, but rather exist as a result of some (unfortunate) decision by a human. For a more technical example of the definitions of fault, error, and failure, consider the following method:

Código-fonte 5: Código contido no exercício de inspeção

```
1 public static int numZero (int[] x) {
2 // Effects: if x == null throw NullPointerException
3 // else return the number of occurrences of 0 in x
4 int count = 0;
5 for (int i = 1; i < x.length; i++)
6 {
7 if (x[i] >= 0)
8 {
9       count++;
10 }
11 }
12 return count;
13 }
```

The error in this program is that it starts looking for zeroes at index 1 instead of index 0, as is necessary for arrays in Java. For example, numZero ([2, 7, 0]) correctly evaluates to 1, while numZero ([0, 7, 2]) incorrectly evaluates to 0. In both of these cases the fault is executed. Although both of these cases result in an error, only the second case results in fault. To understand the error states, we need to identify the state for the program. The state for numZero consists of values for the variables x, count, i, and the program counter (denoted PC). For the first example given above, the state at the if statement on the very first iteration of the loop is (x =[2, 7, 0], count = 0, i = 1, PC = if). Notice that this state is in error precisely because the value of i should be zero on the first iteration. However, since the value of count is coincidentally correct, the error state does not propagate to the output, and hence the software does not fail. In other words, a state is in error simply if it is not the expected state, even if all of the values in the state, considered in isolation, are acceptable. More generally, if the required sequence of states is s0, s1, s2, . . . , and the actual sequence of states is s0, s2, s3, . . . , then state s2 is in error in the second sequence. In the second case the corresponding (error) state is (x=[0, 7, 2], count=0, i=1, PC = if). In this case, the bug propagates to the variable count and is present in the return value of the method.

Hence a failure result.

A practical problem associated with software testing is how to provide the right values to the software and observing details of the software's behavior. A test case is the combination of input value and result necessary for a complete execution and evaluation of the software under test. A test set is a set of test cases and the testing coverage is a property of a test set, rather than a property of a single test case.

No matter what coverage criterion is used, sooner or later one wants to know whether a given program executes correctly on a given input. This is the oracle problem in software testing. The oracle problem can be surprisingly difficult to solve, and so it helps to have a range of approaches available.

One of the aims of testing is to reveal as many failures as possible. Many have been developed to do this (functional, structural, fault-based). These techniques attempt to "break" a program by being as systematic as possible in identifying inputs that will produce representative program behaviors; for instance, by considering subclasses of the input domain, scenarios, states, and data flows. The classification of testing techniques is based on how tests are generated: from the software engineer's intuition and experience, the specifications, the code structure, the real or imagined faults to be discovered, models, or the nature of the application. One category deal with the combined use of two or more techniques. Sometimes these techniques are classified as white-box and black-box:

- **White-box testing:** Deriving tests from external descriptions of the software, including specifications, requirements, and design.
- **Black-box testing:** Deriving tests from the source code internals of the software, specifically including branches, individual conditions, and statements.

Software testing is normally conducted in phases, as software is developed. The three main phases are:

- **Unit Testing:** this phase verifies the functioning in isolation of software elements that are separately testable. Depending on the context, these could be the individual subprograms or a more significant component made of highly cohesive units;
- **Integration Testing:** this phase occurs after the units are tested individually, and aims to ensure that the interactions among software components are working correctly;
- **System Testing:** this phase is concerned with testing the behavior of an entire system, and it is usually considered appropriate for assessing the nonfunctional system requirements such as security, speed, accuracy, and reliability.

A test selection criterion is a means of selecting test cases or determining that a set of test cases is sufficient for a specified purpose. Test adequacy criteria can be used to decide when sufficient testing will be, or has been accomplished.

A well-known functional testing criterion is equivalence partitioning. This criterion involves partitioning the input domain into a collection of subsets (or equivalent classes) based on a specified criterion. This criterion may be different computational results, a relation based on control flow or data flow, or a distinction made between valid inputs that are accepted and processed by the system and invalid inputs, such as out of range values, that are not accepted and should generate an error message or initiate error processing. A representative set of tests (sometimes only one) is usually taken from each equivalency class. The idea is that each test case represents all input domain that equivalency class.

An important aspect is the software techniques are complementary; in fact, they use different sources of information and have been shown to highlight different kinds of problems. They could be used in combination, depending on budgetary considerations.