UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR

INGRID CARDIN DA COSTA
BEATRIZ SILVA LIMA

Sistema WEB de Monitoramento de Textos em Redes Sociais

Prof. Filipe Braida do Carmo, D.Sc.
Orientador

Nova Iguaçu, Abril de 2022

Sistema WEB de Monitoramento de Textos em Redes Sociais

Ingrid Cardin da Costa

Beatriz Silva Lima

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____

Ingrid Cardin da Costa

_____

Beatriz Silva Lima

Aprovado por:

_____

Prof. Filipe Braida do Carmo, D.Sc.

_____

Prof. Natália Chaves Lessa Schots, D.Sc.

_____

Prof. Bruno José Dembogurski, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Abril de 2022

Agradecimentos

Ingrid Cardin da Costa

Muito Obrigada, primeiramente, à Beatriz Silva Lima, minha amiga desde o primeiro período de faculdade e minha dupla para este trabalho. Sendo a pessoa que, desde o inicio, sempre complementou meus conhecimentos, sabendo o que me faltava saber, e sempre chamou atenção para meus erros e falhas, me auxiliando a conserta-los.

Agradeço a meus amigos que sempre me apoiaram, em especial os que conheci durante a faculdade e levarei suas amizades para o resto de minha vida. Dentre eles, principalmente ao Vitor Diniz, que foi praticamente meu coorientador, sempre apresentando dicas e conselhos sobre escrita ou sobre latex.

À minha mãe, Carmen Terezinha de Jesus Tostes dos Santos, que mesmo longe me proporcionou todo o seu apoio, que se provou essencial, sempre se preocupando em perguntar sobre o andamento e sobre como eu me sentia.

Ao meu pai, Flávio Luiz Corrêa da Costa, que sempre se mostrou disposto a me aconselhar sobre tudo que podia. Ficando do meu lado e oferecendo seu apoio.

Ao João Pedro, que nessa fase final de escrita e entregas foi a pessoa que mais me acalmou durante momentos de nervosismo.

Agradeço também a todos os meus professores e membros do Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro, que me proporcionaram conhecimento e diversos tipos de lições, que se provaram de ajuda para este trabalho.

Ao meu orientador Filipe Braida do Carmo, sempre compreensivo e se provando disposto a auxiliar e aconselhar em todos os aspectos deste trabalho, tanto implementação quanto escrita.

Agradeço aos meus colegas de trabalho, que mesmo não conhecendo detalhes sobre meu trabalho de conclusão de curso, sempre ofereceram apoio e conselhos.

Meus sinceros agradecimentos à todos.

Beatriz Silva Lima

Gostaria de agradecer inicialmente à Ingrid Cardin da Costa, minha dupla neste trabalho e uma grande amiga que levo da faculdade para a vida. Ela esteve comigo durante todo o processo de desenvolvimento deste trabalho, me apoiando, ajudando e criticando quando necessário para minha melhora e melhora do nosso trabalho.

Agradeço também meu orientador Filipe Braida do Carmo, que sempre foi compreensivo com minhas dificuldades e disposto a ajudar durante todo o desenvolvimento deste trabalho, desde antes da escolha do tema.

Muito obrigada também à minha mãe, Rita de Cassia Omena da Silva, que sempre tentou me acalmar e me apoiar de todas as formas possíveis durante o processo.

Aos meus amigos que sempre me estiveram por perto, me apoiando e me incentivando a continuar. Dentre eles, um agradecimento especial ao Victor Diniz, que sempre esteve disponível para nos dar dicas e conselhos.

À todos os meus professores e membros do Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro, que me muniram com o conhecimento necessário para chegar até aqui.

Por fim, gostaria de agradecer ao meu pai, Eduardo Ernani Lima, que desde pequena me ensinou que conhecimento é o bem mais valioso que alguém pode possuir.

Meus sinceros agradecimentos à todos.

RESUMO

Sistema WEB de Monitoramento de Textos em Redes Sociais

Ingrid Cardin da Costa e Beatriz Silva Lima

Abril/2022

Orientador: Filipe Braida do Carmo, D.Sc.

Diante de diversos riscos que redes sociais apresentam para a vida pessoal e social, para a saúde mental e para o direito de imagem das pessoas, dois dos grupos mais vulneráveis a estes são o das crianças e adolescentes. Normalmente, muitos jovens ainda não possuem maturidade o suficiente para lidar com riscos ou ameaças que possam vir a sofrer *online*, como por exemplo *cyberbulling*, *phishing*, entre outros. Nesses casos, a existência do aconselhamento dos pais faz toda a diferença na proteção dos jovens, porém, grande parte dos responsáveis pode não saber dos problemas que seus filhos estão tendo *online*. Dado esse quadro preocupante, nosso trabalho tem o objetivo de servir como ferramenta de apoio para a análise de riscos sofridos por jovens em redes sociais famosas, no caso o Twitter. Essa ferramenta funciona analisando os *tweets* e respostas de *tweets* de determinado usuário. Nela basta informar o nome de usuário, e este passará a ser monitorado e analisado. Qualquer *tweet* preocupante que indique sinais de *cyberbulling* será marcado e mostrado para o usuário. Diferentemente de outros trabalhos, para utilização de nosso projeto basta indicar o nome de usuário do indivíduo na rede social em questão, e com isso teremos os resultados da análise e possíveis alertas, sem necessitar de permissões ou de relacionamento comprovado com o indivíduo.

ABSTRACT

Sistema WEB de Monitoramento de Textos em Redes Sociais

Ingrid Cardin da Costa and Beatriz Silva Lima

Abril/2022

Advisor: Filipe Braida do Carmo, D.Sc.

Given the diverse range of risks that social media present to personal and social life, mental health and for image rights, two of the most vulnerable groups are these of children and teenagers. Usually, many young people are not yet mature enough to deal with risks or threats that they may suffer online, such as cyberbulling, phishing, among others. In cases like these, the existence of parental counseling makes all the difference in protecting youngsters, however, a large percentage of parents may not be aware of the problems their kids are having online. With this worrying situation in mind, our work aims to serve as support tool for the analysis of risks suffered by children on common social networks, in the case of this project Twitter accounts. It works by analyzing tweets of a given user and tweets' responses they receive. In the tool, it's necessary to imput the username and the social network in question, and they will be monitored and analyzed. Any worrisome tweet, that indicates signs of cyberbulling for example will be flagged and shown to the user. Unlike other works, in order to use this tool, just type in the username of the individual on the social network in question, and you will recieve an analysis results and possible alerts, without the need for permissions or proven relationship with the individual you want to monitor.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	*Interface de programação de aplicação*
POSTS	*Postagens*
WWW	*World Wide Web*
IE1	*Internet Explorer 1*
WBS	*Web Servers*
RU	*Requisito de Usuário*
RN	*Regra de Negócio*
FP	*Falso Positivo*
FN	*Falso Negativo*
IA	*Inteligência Artificial*
ID	*Número Identificador*
APIs	*Interfaces de programação de aplicação*
NPM	*Node Package Manager*

Sumário

Capítulo 1

Introdução

Vivemos em uma sociedade tecnológica, onde o uso de computadores, celulares e outros meios de comunicação e de acesso à internet estão presentes no dia a dia de uma grande quantidade de pessoas de diversas idades. Consequentemente, nos últimos anos, vem aumentando, além do número de pessoas com idades em geral, o número de crianças e adolescentes utilizando redes sociais como meio de comunicação.

Dentro desse grupo de menores de idade, há muitos que diversas vezes não sabem como agir diante de brechas na segurança, riscos e ameaças sofridas *online*, de maneira direta ou indireta. Esse grupo requer a devida atenção de seus responsáveis, tanto por merecerem seu auxílio, quanto por também necessitarem de aconselhamento e proteção diante de algum perigo.

Diante deste cenário, riscos como *cyberbullying*, roubo de informações pessoais, *grooming*, exposição a conteúdos inadequados, entre outros, que já existiam para todas as idades são ainda mais agravados para esse grupo, já que uma grande porcentagem de menores não tomam devidas precauções de segurança ao utilizarem as redes sociais.

Apesar da necessidade da proteção e aconselhamento dos pais em suas atividades *online*, muitas crianças e adolescentes não recebem esse tipo de informação. Para muitos responsáveis é inviável monitorar as atividades dos menores, devido à falta de tempo ao ter que conciliar este monitoramento com trabalho e vida pessoal. Há

também a vontade de alguns pais de respeitar a privacidade de seus filhos, juntamente ao medo de se intrometer demais na vida pessoal dos filhos e afastá-los.

1.1 Objetivo

Apesar do respeito à privacidade, ainda é de extrema importância a presença e aconselhamento dos pais, principalmente em situações que oferecem perigo aos filhos ou em situações onde eles deverão ser aconselhados ou desencorajados. Visando auxiliar os pais e responsáveis nesse quesito, o objetivo deste trabalho é desenvolver uma ferramenta de detecção de ameaças recebidas ou enviadas por usuários da rede social Twitter[1], através da análise das mensagens trocadas com outros usuários.

Este trabalho consiste em um sistema para *Web*, a partir do qual um usuário pode monitorar as mensagens de certos usuários na rede social Twitter. Este monitoramento apresenta-se na ferramenta a partir de um *dashboard*, onde são exibidas todas as informações necessárias dos indivíduos sendo monitorados, como nome, últimas mensagens e possíveis alertas.

A classificação das mensagens como preocupantes ou não foi feita através do uso de um dicionário, onde através da presença ou não de determinadas palavras a mensagem é classificada ou não como um alerta. Além disso, o projeto foi feito utiizando o *framework* AdonisJS, juntamente com o *framework frontend* Tailwind CSS. Utilizamos a API do Twitter para a coleta dos tweets dos usuários da rede social.

1.2 Organização do Trabalho

O trabalho está organizado em cinco capítulos, consistidos em uma introdução, fundamentação, proposta, experimentos e conclusão. A seguir, está o detalhamento de cada capítulo.

[1]https://twitter.com/home

- Capítulo 1: Consiste em uma pequena e concisa introdução geral do projeto e seu contexto.

- Capítulo 2: Consiste na fundamentação do trabalho. Nele explicaremos os conceitos técnicos básicos necessários para o entendimento do trabalho, além de apresentarmos uma pequena introdução da história da *Web*, para melhor compreensão do surgimento das ferramentas utilizadas hoje em dia.

- Capítulo 3: Contém a proposta e a motivação do trabalho. Compõe-se da parte que apresenta e discorre sobre o problema social da falta de displicência acompanhada do uso da Internet por parte dos menores de idade, que se tornou a motivação para a criação desse trabalho. Além disso, também apresentamos e recorremos sobre alguns trabalhos relacionados sobre o assunto e *apps* e softwares funcionalidades semelhantes ao nosso sistema. Por último, apresentamos o planejamento da implementação do trabalho.

- Capítulo 4: Demonstra como o projeto foi desenvolvido, apresentando as tecnologias utilizadas para a implementação do sistema. Além disso, também descreve e explica o sistema em geral, sua arquitetura e implementação.

- Capítulo 5: Apresenta um breve resumo do que foi feito no presente trabalho, limitações encontradas para a pesquisa e implementação, possíveis extensões e, por fim, considerações finais.

Capítulo 2

Fundamentação

Neste capítulo falaremos brevemente sobre a história do desenvolvimento *Web*, e toda a base teórica necessária para o entendimento dos *frameworks* utilizados para a construção do trabalho.

2.1 Rede Mundial de Computadores

O primeiro passo para o surgimento do que conhecemos hoje como a Internet ou Rede Mundial de Computadores (*Wo rld Wide Web* - WWW) ocorreu com a construção da *ARPANET*[1] em 1969 (PALOQUE-BERGèS; SCHAFER, 2019). Ela consistia em uma rede de computadores que ligava diferentes sistemas computacionais com o intuito de transmitir dados. Seu objetivo era interligar departamentos de pesquisa e transmitir informações militares sigilosas. Segundo Paloque-Bergès e Schafer (2019), a primeira conexão entre dois computadores ocorreu em 1969 em um laboratório da *UCLA*[2] que ficava localizada na Universidade da Califórnia em Los Angeles, nos Estados Unidos. Além disso, os autores também contam que com a Arpanet se deu a primeira conexão ponto a ponto entre um computador da UCLA e um da universidade de Stanford, também nos Estados Unidos.

Após isso, em 1989, o Britânico Tim Berners-Lee criou, para a empresa CERN,

[1] <https://developer.mozilla.org/pt-BR/docs/Glossary/Arpanet>
[2] <https://www.ucla.edu/>

uma proposta que descrevia um projeto chamado *World Wide Web*, onde uma teia de documentos hipertextos interligados poderiam ser visualizados por navegadores. O objetivo desse projeto era, em um primeiro momento, auxiliar no compartilhamento de informações entre cientistas e pesquisadores de diferentes instituições, que sofriam com a falta de um meio de comunicação rápido e sem fio. A ideia da WWW o permitiu criar, através de protocolos *web*, o primeiro navegador e também o primeiro servidor *web*, que é um computador que armazena os arquivos que compõem um site. Desse modo, o compartilhamento se baseava em uma conexão entre navegador e servidor (BERNERS-LEE, 2000).

A WWW não foi algo originado subitamente. Cerca de duas décadas antes, diversos pesquisadores já criavam tecnologias que serviriam como base para a WWW. Como o próprio Tim Berners-Lee diz em seu livro *Weaving The Web* (BERNERS-LEE, 2000), a WWW foi resultado de diversas influências, de conversas, descobertas e de experimentos sem ligações uns com os outros, que interligaram-se em sua cabeça.

O primeiro navegador *web*, criado por Tim Berners-Lee em 1990 juntamente com o primeiro servidor, foi chamado de WorldWideWeb e depois renomeado para Nexus, para evitar confusões com a *World Wide Web*. Em um primeiro momento, ele foi o único meio de acesso à *web*. Em 1992, *logs* do primeiro servidor *web* mostravam um aumento exponencial em seu uso, o que acarretou no fato de diversos engenheiros, cientistas e estudantes começarem a trabalhar na criação de novos navegadores, cada um deles tentando incluir cada vez mais novas funcionalidades para diferenciar seus trabalhos de seu precursor (BERNERS-LEE, 2000). Hoje em dia, o acesso a esse navegador pode ser encontrado na página do projeto WorldWideWeb (CERN, 1990). Ele possui uma aparência bem simples, sendo sua página inicial mostrada na figura 2.1.

Mark Andreessen e Eric Bina, dois membros do corpo acadêmico do NCSA (Centro Nacional de Aplicações de Supercomputação da universidade de Illinois) que fica localizado no EUA, se juntaram para fazer um novo navegador, que denominaram de *Mosaic*[3]. Ele foi disponibilizado para o público geral em 1993 com foco em ser

[3]<http://www.ncsa.illinois.edu/enabling/mosaic>

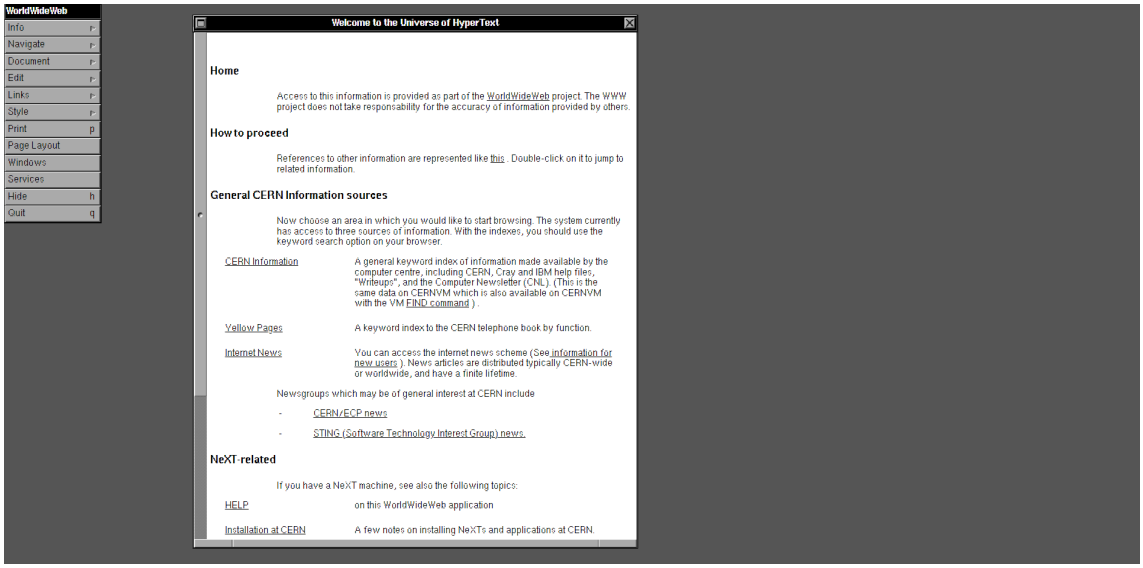

Figura 2.1: Página inicial do projeto WorldWideWeb[4].

mais simples de instalar e utilizar, tornando-se popular na época. Além disso, esse navegador foi um dos primeiros a permitir hipermídia, que é uma mídia de leitura que pode conter, não só hipertexto, como também imagens e sons.

Um ano após a disponibilização do Mosaic, também houve a sua descontinuação. Em 1994, Andreessen participou da fundação da empresa Netscape Communications Corporation[5], o que acarretou na criação do Netscape Navigator, que prometia ser um melhor navegador que o anterior em termos de velocidade.

Outro famoso navegador influenciado pelo *Mosaic* foi o Internet Explorer[6]. Sua primeira versão, anteriormente chamada de *Microsoft Internet Explorer* ou de *Internet Explorer 1* (IE1), foi lançada em 1995 pela Microsoft[7]. Ele está sendo utilizado até os dias de hoje, porém em suas versões mais recentes, e atualmente já se encontra na versão 11. Porém já foi confirmado pela Microsoft que essa será sua última versão e que a partir de junho de 2022 deixará de receber suporte em certas versões do Windows 10[8].

[4]Imagem retirada do projeto Worldwideweb no endereço: <https://worldwideweb.cern.ch/browser/>

[5]<https://www.britannica.com/topic/Netscape-Communications-Corp>

[6]<https://docs.microsoft.com/pt-br/internet-explorer/internet-explorer>

[7]<https://www.microsoft.com/pt-br>

[8]<https://docs.microsoft.com/pt-br/lifecycle/announcements/internet-explorer-11-end-of-support>

Com o lançamento do Internet Explorer 1, famosos navegadores continuaram a ser desenvolvidos, em um fenômeno que foi chamado de *Browser Wars*. Windrum (2004) explica que com o lançamento do IE1, a companhia Netscape continuou criando novas versões de seu navegador em paralelo a Microsoft, ambas tentando superar uma a outra. Nesse período, outros navegadores conhecidos também foram lançados, como o *Opera*[9] em 1996 e o *Safari*[10] em 2003. Com o declínio do uso de Netscape, a *Mozila*[11] lançou o *Firefox*[12] em 2004. Além disso, a *Google*[13] desenvolveu o *Google Chrome*[14] em 2008, e a Microsoft lançou o *Microsoft Edge*[15] em 2015.

O surgimento e evolução de novos navegadores foi diretamente ligado à evolução da Internet e de seu conteúdo como um todo, o que também se relaciona ao grande aumento no número de sites e de ferramentas utilizadas para sua criação. Na próxima sessão, falaremos um pouco sobre a base teórica do desenvolvimento *Web*.

2.2 Arquitetura Web

O desenvolvimento *web* se baseia na criação e manutenção de sites e páginas da *Web*, que são um conjunto de documentos codificados e transformados no que vemos pelo navegador. Uma página da *Web* pode ser classificada basicamente em duas partes: o lado do cliente e o lado do servidor. Ademais, existem dois principais tipos de conexões entre dois computadores: *peer-to-peer*, que não diferencia clientes de servidores, e cliente-servidor, que diferencia entre computadores com papel de cliente e computadores com papel de servidor.

[9] <https://www.opera.com/pt-br/browsers>
[10] <https://www.apple.com/br/safari/>
[11] <https://foundation.mozilla.org/pt-BR/?utm_source=www.mozilla.org&utm_campaign=nav&utm_medium=referral&utm_content=who-we-are>
[12] <https://www.mozilla.org/pt-BR/firefox/new/>
[13] <https://www.google.com/intl/pt-BR_br/business/>
[14] <https://www.google.com/intl/pt-BR/chrome/browser-tools/>
[15] <https://www.microsoft.com/pt-br/edge>

2.2.1 Servidores Web

Páginas *Web* são conjuntos de hipermídia (como são chamadas mídias de diversos tipos no meio computacional) que podem conter diversos documentos como textos, textos formatado utilizando alguma linguagem de marcação como o HTML (Linguagem de Marcação de HiperTexto), documentos de estilização como o CSS (Folhas de Estilo em Cascata), e arquivos com funções e *scripts* Javascript para páginas da *Web* interativas. Um computador que hospeda, processa e disponibiliza estes documentos e arquivos é o que chamamos de servidor *web*. O *web server* possui, além dos arquivos armazenados, um software que processa os *requests* que chegam nele, utilizando o padrão HTTP para a transferência de hipermídia. O HTTP (Protocolo de Transferência de Hipertexto) é um protocolo para a transferência de hipertexto, ou seja, ele envia como resposta um conjunto de arquivos que serão transformados em uma página pronta para visualização no computador que enviou o *request* ao servidor. Este computador que recebe a resposta é chamado de cliente.

A resposta que um *servidor Web* envia ao computador cliente, de modo geral, é composta de um arquivo HTML, podendo também possuir caminhos para outros arquivos, *e.g.* estilos em CSS, *scripts* escritos em Javascript, vídeos, imagens, arquivos de áudio e documentos de texto. O navegador é um aplicativo que tem o objetivo de interpretar os arquivos HTML e transformá-los em algo visual, que é a pagina *Web* que vemos.

A primeira conexão entre dois computadores aconteceu com a Arpanet, e foi uma conexão ponto a ponto, ou seja, uma conexão entre dois computadores, onde ambos tinham as mesmas funcionalidades, tanto de cliente como de servidor. Isso significa que ambos enviavam um pacote de dados definido pelo protocolo HTTP que é chamado de *request* e recebiam *responses*, que é a resposta do servidor, como demonstra a Figura 2.2.

Junto com a WWW, também veio a conexão cliente-servidor, representada pela Figura 2.3, onde um computador seria o “cliente” e o outro o “servidor”, sendo o servidor um *web server*. Já o cliente era o que enviava os *requests* e recebia os

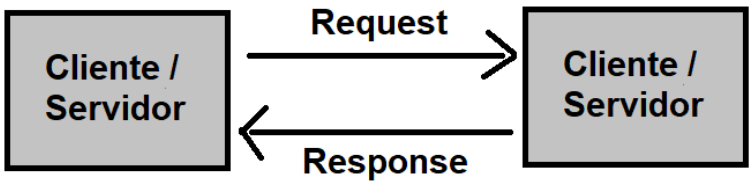

Figura 2.2: Conexão entre dois computadores, ambos exercendo papel de cliente e de servidor.

documentos.

Figura 2.3: Conexão entre computadores com papel de cliente e computadores com papel de servidor.

2.2.2 Protocolos

Os protocolos *web* foram criados com o objetivo de facultar a comunicação entre dois diferentes dispositivos, mesmo que haja uma grande distância entre eles. Tais protocolos definem como os dados trafegam pela *Web* e antes deles, redes de diferentes topologias tinham a necessidade de passar suas mensagens por um computador .

Em 1973 os cientistas Robert Kahn e Vinton Cerf desenvolveram o protocolo TCP/IP(protocolo de controle de transmissão/protocolo da internet), com o objetivo de permitir que qualquer sistema se comunicasse com qualquer outro sistema de qualquer topologia de rede (HALL, 2000), propiciando por exemplo, que um celular *Android*[16] conecte-se com um computador *Linux*[17]. Dez anos depois, em 1983, a Arpanet adotou o TCP/IP e prontamente a Universidade da Califórnia, em Berkeley nos Estados Unidos, também começou a incluir o protocolo no sistema operacional *UNIX*, que é distribuído gratuitamente e bastante utilizado pela comunidade acadêmica na época (HALL, 2000).

[16] <https://www.android.com/intl/pt-BR_br/what-is-android/>
[17] <https://www.linux.org/pages/download/>

2.2.3 HTML, CSS e Javascript

O HTML, *hypertext Markup Language* (linguagem de marcação de hypertexto), é uma linguagem de programação que possui o objetivo de estruturar uma página *Web*. Ele funciona a partir de *tags* que são elementos, ou seja palavras reservadas da linguagem, entre os símbolos "<" e ">". As *tags* indicam o início e o fim de cada elemento. Por exemplo, para criar o título de uma pagina *Web*, faríamos o indicado no código à esquerda da Figura 2.4, e esse código geraria uma página como a indicada na direita da figura.

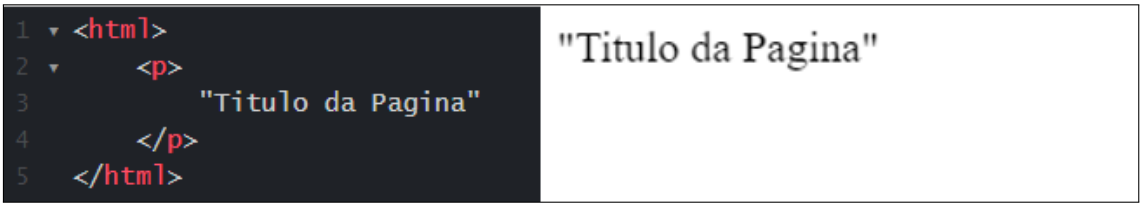

Figura 2.4: Exemplo de código HTML à esquerda, e página gerada por ele à direita

Dentre as *tags* existentes no HTML, há diversos tipos. A *tag* <html>, por exemplo, indica o início de qualquer documento html, já as *tags* <head>, <body> e <footer>, por exemplo, indicam diferentes seções de uma página *Web*. Há também múltiplas outras *tags* como <p> que indica paragrafo, <img>, <audio> e <video> que adicionam imagens, áudios e vídeos respectivamente, e até que é um *container* utilizado para agrupamento de elementos e <nav> que é uma seção da página que aponta para outras seções, ou até mesmo para outras páginas como um *link*.

Com a popularização de páginas *Web* durante a época da guerra dos navegadores, foi surgindo também a necessidade de embelezar essas páginas e para isso existia o elemento <style> no HTML (POWERS, 2012). Esse elemento engloba as informações de estilo de um elemento ou do documento HTML por inteiro, e essas informações de estilo podem ser cor, tamanho, estilo de fonte, entre outras. Porém, paginas *Web* continuaram com sua evolução, obstando o uso da *tag style* como único meio de variação de aspecto da página, e ensejando o surgimento do CSS(POWERS, 2012).

O CSS (*Cascading Style Sheets*) é uma linguagem de marcação geralmente utilizada junto com o HTML. Dentre as diferenças entre seu uso e o da *tag style*, estava o fato de que com ela a possibilidade de estilização utilizando menos linhas de código e menos complexidade é muito maior. Como bem aponta Powers (2012), ao contrário do estilo de programação de páginas que consistia em aplicar estilos em elementos individualmente através do HTML, ao acrescentar o CSS, a programação de estilo fica em um diferente arquivo, auxiliando assim no desacoplamento e na simplicidade do código, o que por sua vez auxilia na redução do tempo necessário para a criação de mais páginas.

A partir da inclusão do arquivo CSS em um projeto, em um arquivo separado, as mudanças de estilo são feitas em um só lugar, e são aplicadas automaticamente em todos os elementos correspondentes de todas as páginas, deixando tudo em um mesmo estilo de maneira mais fácil e mais uma vez provando a diminuição na complexidade. Porém, isso não impede que mudanças sejam feitas separadamente para cada elemento, já que isso também é suportado pelo CSS. Além disso, vale ressaltar que ele também é compatível com alguns navegadores mais antigos, provando assim que, apesar de navegadores não interpretarem o código de uma página de maneira universal, eles são capazes de aceitá-lo (POWERS, 2012).

Outro ponto importante ressaltar quando se fala de CSS é o conceito de Seletores[18]. O arquivo deste formato não só modifica o estilo de uma página ou site inteiros ou de elementos únicos separadamente, como também de um conjunto de elementos relacionados uns com os outros, independente da localização deles na página ou site, e isso é possível graças aos Seletores. Eles servem para indicar quais elementos do HTML determinada regra de um CSS se aplica (POWERS, 2012).

Utilizar o CSS envolve avisar ao navegador instruções com sempre o mesmo formato, indicando primeiramente o Seletor seguido de uma propriedade e um valor. Essa linguagem de marcação proporciona, como meio de amparo aos programadores, propriedades com nomes bem intuitivos, como "*font-size*" para tamanho, "*margin-top*" para um espaço em branco no topo do elemento, "*border-color*" para cor das bordas,

[18] <https://developer.mozilla.org/pt-BR/docs/Web/CSS/CSS_Selectors>

entre outros, todas seguidas de um valor (POWERS, 2012). Os valores podem estar em diversos formatos, designadamente *pixels*, altura, largura e porcentagem para medir tamanhos, como também RGB(sistema de cor, baseado nas cores vermelho, verde e azul), nomes pré definidos de cores ou hexadecimal como valor para uma cor.

Um código CSS não é utilizado sozinho, e sim em conjunto com um HTML. No código apresentado à esquerda da Figura 2.4, podemos ver um exemplo de criação de um parágrafo utilizando um HTML, e na direita da figura como seria a página gerada por esse código. No código à esquerda da Figura 2.5 estamos modificando a cor e adicionando uma margem de 100 *pixels* e uma indentação em um seletor p. Isso significa que, para todo novo paragrafo p criado em nosso HTML, essas configurações serão aplicadas. Um exemplo de como ficaria uma página gerada ao adicionar este trecho de código se encontra na direita da Figura2.5.

Figura 2.5: Exemplo de código CSS a esquerda, e página gerada página a direita.

Com a combinação do HTML e CSS a *Web* já possuía páginas funcionais e com estilos próprios. Porém, um detalhe faltante era o dinamismo nas páginas e por essa razão se dá o uso da linguagem Javascript. Essa linguagem foi desenvolvida pela Netscape antes da época da guerra dos navegadores. Sua sintaxe é baseada na linguagem Java, porém, apesar da sintaxe, não há nenhuma outra semelhança. Além disso é uma linguagem de alto nível, baseada em objetos, dinâmica, interpretada e não tipada (FLANAGAN, 2011).

Uma das influências para a criação do Javascript foi a vontade de enviar requisições de forma assíncrona, ou seja, não ter mais a necessidade de recarregar uma página para enviar uma requisição e receber sua resposta. Ou seja, tendo uma página que é gerada, roda no cliente e busca requisições tudo dinamicamente. Javascript é uma linguagem de *script*, ou seja, que pode dar instruções que permitem que

uma página realize diferentes funções em tempo de execução. Segundo Flanagan (2011), Javascript tornou-se uma linguagem de uso geral robusta e eficiente, inclusive podendo definir novos recursos para desenvolvimento de software em grande escala.

O Javascript foi criado inicialmente para ser usado em pequenas partes do código, não muito complexo. Porém seu uso foi se popularizando cada vez mais, passando a ser usado em projetos cada vez maiores ou mais complexos, acarretando em sua iminente otimização e adicionamento de APIs na linguagem, o que aumentou ainda mais seu uso. Após isso, sua disseminação foi tão grande que passou a ser usado também na criação de *servers*, como por exemplo o node.js.

Capítulo 3

Proposta

É de extrema importância ressaltar a diferença que o aumento no uso de redes sociais acarreta na vida da população em geral. Este aumento também está presente em grupos mais específicos, como por exemplo crianças e adolescentes. Porém, a verdade inquietante desta situação, se dá no fato de que este grupo no geral não possui as melhores noções de segurança *online*, e é um grupo necessitado de cuidado e apoio. Dada a natureza desse problema vale salientar que também existem diversas tentativas de solucioná-los.

Este capítulo demonstra e discorre sobre um dos problemas agravados pela popularização do uso de redes sociais por menores de idade: a questão da falta de segurança de menores *online*. Além disso, também propõe um sistema para auxílio no monitoramento não invasivo de menores de idade, e enuncia e descreve sobre trabalhos e aplicativos com funções parecidas ou relacionadas ao sistema em questão.

3.1 Motivação

A popularização de celulares e computadores residenciais fez com que a Internet se tornasse acessível à população comum, o que resultou em um grande aumento de pessoas se conectando a Internet. Segundo Tirumala, Sarrafzadeh e Pang (2016), entre os anos de 2000 e 2016, o uso global de Internet cresceu 900.4%. Hoje seis a cada

dez pessoas no mundo utilizam a Internet no seu dia a dia (KEMP; HOOTSUITE; SOCIAL, 2021a).

Essa nova ferramenta causou um grande impacto na sociedade, mudando, por exemplo, como as pessoas se comunicam, buscam informação e se expressam. Otimizando essas e muitas outras atividades, tornou-se parte importante do dia a dia das pessoas. De acordo com Kemp, Hootsuite e Social (2021b), atualmente as pessoas passam em média seis horas e cinquenta e oito minutos diários conectados à Internet.

É inegável que a Internet mudou de forma positiva a vida das pessoas. Porém, não só indivíduos de boa índole se adaptaram a esse novo estilo de vida, criminosos também aderiram a ele, criando e adaptando golpes para o mundo virtual. Todas as pessoas que utilizam a Internet estão suscetíveis a ataques como roubo de identidade, *phishing* (crime de enganar pessoas para que compartilhem dados confidenciais) e roubo de dados pessoais. No entanto, alguns grupos são mais vulneráveis a ataques do que outros. Um desses grupos é o de crianças e adolescentes.

Alguns dos motivos do agravamento da vulnerabilidade desse grupo são a falta de consciência de que correm perigo ao navegarem na Internet e falta de conhecimento de como se proteger. Segundo Tirumala, Sarrafzadeh e Pang (2016), apenas 19% dos estudantes entre 8 e 12 anos de idade entrevistados na Nova Zelândia foram capazes de responder perguntas sobre segurança na Internet. Esta taxa é um índice alarmantemente baixo.

Dentre as atividades realizadas por crianças e adolescentes, a que oferece mais perigo, de acordo com Kahimise e Shava (2019), é o uso de redes sociais. Isso porque nesses sites eles compartilham informações pessoais e conversam com estranhos, sem a noção de que aquela pessoa pode ser uma ameaça. Ainda segundo os autores, crianças com baixa autoestima ou com um relacionamento familiar ruim correm ainda mais risco, podendo compartilhar fotos ofensivas a eles ou a seus amigos na busca por aceitação.

No entanto, nem sempre as crianças são as vítimas. Existem cenários nos quais elas se tornam os agressores. Um deles é o do *cyberbullying*, onde uma pessoa ou um

grupo, por meio de dispositivos eletrônicos, agride de forma intencional e repetitiva um indivíduo. É uma variação do *bullying* que acontece nas escolas e, por vezes, uma extensão do mesmo.

A grande recorrência do cenário citado e as graves consequências para a vítima, fizeram com que o *cyberbullying* se tornasse assunto de pesquisas, discussões e até mesmo séries e filmes, que procuram disseminar conscientização sobre o assunto e diminuir o número de casos. De acordo com Chan, Cheung e Lee (2021), em uma pesquisa feita com adolescentes nos Estados Unidos, 33.8% dos entrevistados disseram ter sofrido com o *cyberbullying* e 11.5% confessaram ter sido agressores em alguma ocasião.

Formas de mitigar os ataques a crianças e adolescentes na Internet vem sendo discutidas na literatura. Uma dessas medidas é a de promover conscientização sobre a segurança na Internet às crianças e adolescentes, tornando-os mais aptos a escolher de forma mais apropriada quais informações são pessoais demais para serem compartilhadas e com quem é seguro compartilhá-las.

É possível, porém, que mesmo com o conhecimento sobre os possíveis riscos, crianças e adolescentes ainda exibam comportamentos que os exponham ao perigo. Segundo Vanderhoven et al. (2014), mesmo após a conscientização dos adolescentes que participaram da pesquisa não houve mudança de comportamento dos participantes nas redes sociais. O que evidencia que conscientização não é o suficiente para proteger esse grupo.

Outra abordagem é envolver os pais ou guardiões desses jovens em sua vida virtual. Essa medida é mais efetiva pois o adulto age, não só como um regulador do comportamento dos jovens, supervisionando as informações que compartilham e com quem se comunicam, como também uma barreira, que monitora e protege as crianças e adolescentes de ataques vindos de outras pessoas.

Todavia, é importante ressaltar que, para que a intervenção dos pais seja benéfica é preciso que ela seja de caráter de apoio à criança, mantendo ao máximo a liberdade e autonomia da mesma. De acordo com Ghosh et al. (2018), um estilo parental

autoritário se mostrou relacionado com o aumento das chances do filho sofrer com *cyberbullying* e ter problemas com seus amigos. O autor também aponta que as restrições podem fazer com que o adolescente seja excluído por seus colegas.

Segundo Ang (2015), a falta de ligação entre pais e filhos e a falta de monitoramento dos pais nas atividades virtuais dos filhos estão ligadas a ocorrência de *cyberbullying*. Além disso, esses pontos são alvos adequados na construção de soluções para a prevenção e intervenção afim de impedir o *cyberbullying*. Isso mostra que a supervisão e cuidado, ao contrário de restrição e autoritarismo, são chaves importantes para a obtenção de um ambiente virtual mais seguro para crianças e adolescentes.

Considerando esse contexto, a proposta deste trabalho é construir uma ferramenta que auxilie os pais a monitorarem a vida social virtual de seus filhos, sendo avisados de possíveis ameaças a seus filhos ou condutas ofensivas que eles tenham nas redes sociais. Desta forma os pais poderão supervisionar com mais eficiência e tomar as medidas cabíveis em decorrência de algum evento que represente algum risco para os seus filhos ou para as crianças e adolescentes com quem seus filhos se relacionam nas redes sociais.

3.2 Trabalhos Relacionados

Nesta seção apresentaremos e articularemos sobre artigos e trabalhos que apresentam soluções para o monitoramento de crianças e adolescentes nas redes sociais. Esses trabalhos tratam e propõem soluções para ameaças que os jovens sofrem *online*.

Uma das redes sociais mais utilizadas ao redor do mundo é o *Facebook*[1]. Muitas crianças e adolescentes o utilizam, fazendo postagens, comentando, conversando e conhecendo novas pessoas. O *Facebook* fornece um controle limitado para pais, onde eles, para editarem as configurações de privacidade e acompanharem o registro de atividades dos filhos, precisam estar utilizando a conta dos mesmos. Além disso, todas as ações do registro de atividades devem ser checadas manualmente, o que de

[1] <https://www.facebook.com/>

certa forma não é o ideal, tanto pela grande quantidade de postagens, quanto pela vontade de manter a privacidade dos filhos.

Outras redes sociais muito utilizadas também se encontram na situação mencionada anteriormente. Em comentários e respostas em fotos do *Instagram*[2] e em vídeos no *Youtube*[3], assim como discussões no *Twitter*[4] e em fóruns da Internet, por exemplo, não existe um sistema que identifique e controle possíveis ameaças para crianças e adolescentes, e assim como nas atividades do *Facebook*, é inviável aos pais controlarem manualmente tudo o que seus filhos escrevem e leem nestas redes sociais. Com essa preocupação em mente, muitos trabalhos, pesquisas e aplicativos foram desenvolvidos como soluções.

Menini et al. (2019) propõem um sistema para monitorar casos de *cyberbulling* baseado na combinação de análise de redes sociais e classificação de mensagens, focado em análises do *Instagram*. O algorítimo do sistema inicia classificando grupos de pessoas como comunidades, e faz isso, primeiramente, encontrando pessoas que costumam postar nos mesmos lugares, e dessas, quais tem contatos umas com as outras. Após isso, segue com a classificação das mensagens dos usuários, sendo feita separadamente para cada grupo. Para fazer a classificação se determinada mensagem contém ou não conteúdo abusivo, foi utilizada uma arquitetura neural modular para classificação em uma API de aprendizado de maquina chamada Keras[5], que se provou ser efetiva e robusta para o trabalho segundo os autores.

Já em Informatics et al. (2015), é proposto e implementado um aplicativo para celulares *Android*[6], que auxilia no monitoramento de atividades de crianças, adolescentes e pré-adolescentes no *Facebook*. Este trabalho foi dividido em duas partes, onde a primeira se baseou na criação de uma simulação de uma rede social baseada no *Facebook*. Essa simulação foi feita para ser utilizada como base de testes, pois, para utilizar uma API no *Facebook* que acesse páginas de perfil e *feeds* de notícias, são necessárias diversas permissões que os autores não puderam adquirir. A segunda

[2] <https://www.instagram.com/>
[3] <https://www.youtube.com/>
[4] <https://Twitter.com/home>
[5] <https://keras.io/about/>
[6] <https://www.android.com/>

parte consiste na aplicação feita com o *Android* SDK[7]. Uma parte importante da aplicação foi a parte da autenticação, onde, antes do uso do aplicativo, os mentores deveriam comprovar através da documentação necessária, o seu grau de parentesco com o jovem. A API salva em um banco de dados qualquer *input* de usuário que for classificado como postagem, respostas em outras postagens, mensagens enviadas e mensagens respondidas, incluindo data e hora na qual cada mensagem e postagem ocorreu. Dado isso, a API envia essas informações para a aplicação *mobile* assim que necessário. Desse modo, os responsáveis podem analisar as atividades de seus filhos na data e hora que quiserem caso alguma preocupação surja.

O software NetNanny (2018) é um dos softwares de monitoramento mais conhecidos e utilizados nos Estados Unidos, sendo analisado por diversos programas televisivos e canais de notícias como *ABC News*[8], *NBC*[9] e *Today*[10]. Sendo usado desde 1996, o *Net Nanny* utiliza inteligência artificial para analisar cada conteúdo de uma página *Web* assim que o *link* é clicado. A partir disso, o sistema bloqueia a página caso nela seja encontrado algum conteúdo inapropriado, como pornografia ou material adulto, além de notificar os responsáveis. Assim como outros aplicativos existentes, o *Net Nanny* também oferece a possibilidade de limitar o tempo de uso de redes sociais e bloquear aplicativos e *websites* específicos.

O Bark[11] é um aplicativo de monitoramento de crianças e adolescentes nas redes sociais mostrado na figura 3.1. Ele faz isso de maneira pouco invasiva para a privacidade, monitorando as atividades dos jovens e classificando-as caso haja alguma ameaça. Além disso, o aplicativo reporta somente o perigo identificado e o conteúdo em questão para os pais. O monitoramento do aplicativo é feito em níveis ajustáveis, cabendo aos pais decidirem o que será monitorado e quando o monitoramento ocorrerá. O Bark pode monitorar celulares, computadores, conteúdo de *e-mails*, *apps* e navegadores como o *Google Chrome* e *Microsoft Edge*. Entre os *apps* que podem ser monitorados encontram-se as mais famosas redes sociais. Sendo

[7] <https://developer.android.com/studio>
[8] <https://abcnews.go.com/>
[9] <https://www.nbcnews.com/>
[10] <https://abcnews.go.com/>
[11] <https://www.bark.us/>

assim, são analisados posts e comentários do *Facebook*, *Reddit*, *Twitter*, conteúdo e comentários de vídeos do *Youtube*, letras de músicas ouvidas no *Spotify*, entre outras.

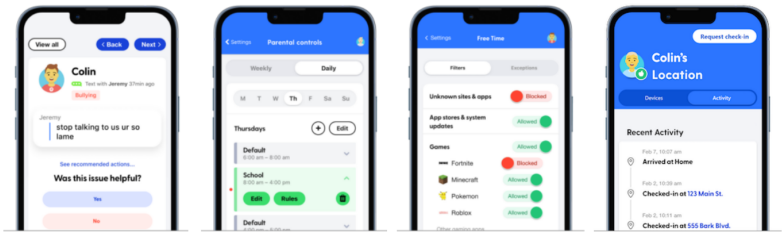

Figura 3.1: Algumas Funcionalidades do Aplicativo Bark[12]

Outro sistema com finalidades semelhantes as do Bark, é o OurPack[13]. O software em questão está disponível para celulares *Android* e *iOS*[14] e possui versões pagas e gratuitas. O aplicativo possui diversas funcionalidades como disponibilizar a localização do dispositivo da criança a qualquer momento, setar limites de tempo e horário em que a criança pode utilizar a Internet e redes sociais. Ademais, pode bloquear e liberar manualmente o acesso a cada aplicação presente no dispositivo. Na versão *premium*, é permitido que até 20 dispositivos sejam controlados simultaneamente, além de incluir funcionalidades de bloqueio automático para horários determinados e um filtro para impedir acesso a conteúdo adulto.

3.3 Proposta

Como apresentado, a detecção automática de ameaças para crianças se torna uma necessidade no dia a dia dos pais, principalmente dos que necessitam dividir seu tempo em diversas atividades. Por isso, trabalhos e aplicativos que tenham esta finalidade se tornam extremamente necessários, o que acarreta na grande diversidade de trabalhos semelhantes existentes, e também na grande procura por parte de responsáveis. Esses aplicativos não existem com a finalidade de invadir a privacidade dos jovens, ou vigiá-los, e sim como apoio para casos onde o jovem sofra algum tipo de ameaça e não perceba ou não consiga procurar por ajuda sozinho.

[12]Imagem retirada da pagina inicial site do aplicativo, na url: <https://www.bark.us/>, e acessada por último no dia 13/04/2022
[13]<https://ourpact.com/>
[14]<https://www.apple.com/br/iphone/>

O intuito deste trabalho é criar um sistema de detecção de ameaça, *e.g.* assédio virtual, *phishing* e pedofilia, no Twitter. O sistema permite que, após cadastrar uma nova conta e realizar o login, um usuário possa monitorar perfis do Twitter a partir do nome de usuário. Ao escolher um nome de usuário, o sistema inicia o monitoramento do usuário, e marca os *tweets* preocupantes. Somente *tweets* e respostas, tanto do usuário quanto direcionadas a ele, com *flags* suspeitas serão mostrados, possibilitando assim, sem grande uso do tempo dos pais, controle e conhecimento de atividades suspeitas relacionadas aos filhos na rede social.

A parte de monitoramento do trabalho consiste em um *dashboard*, onde aparecerão as opções para começar e parar o monitoramento de um usuário, além de também mostrar as informações de um usuário que está sendo monitorado no momento. As informações do usuário consistem no nome de usuário na plataforma em que ele está sendo monitorado, no caso seu nome no Twitter, sua foto de perfil da plataforma, seus últimos *tweets* e número de alertas. Também contém no *dashboard* o *status* de cada alerta. Este *status* indica se um alerta foi ou não visto ou resolvido pelo usuário.

Ao reaver as mensagens de um usuário do Twitter, são armazenadas todas as mensagens dos últimos sete dias relacionadas a esse usuário, tanto seus *tweets* quanto respostas a *tweets* de outros usuários e de seus próprios *tweets*. Além disso, a recuperação de mensagens ocorrerá automaticamente a cada 15 minutos, através de um *cronjob*. O *cronjob* se baseia em uma ferramenta com o trabalho de controlar a execução de uma mesma tarefa repetidamente, e em um tempo especificado de antemão. Ou seja, no caso deste trabalho, a cada ciclo de tempo de 15 minutos todas as mensagens disponíveis para resgate são salvas e analisadas com a utilização de um dicionário.

É importante ressaltar que este projeto deverá possuir uma arquitetura expansiva, já que um de nossos objetivos futuros é melhorá-lo e acrescentar novas redes sociais, como Instagram[15] e LinkedIn[16], e novas formas de classificação de mensagens, como utilização de técnicas de IA.

[15]https://www.instagram.com/
[16]https://www.LinkedIn.com/feed/

3.3.1 Falsos Alertas

Dada a nuância presente na língua portuguesa durante conversas e frases coloquiais, determinadas palavras com intuito negativo podem ser usadas em uma frase que não compactua da mesma disposição. Casos esses como em frases onde podem estar presentes a ironia, exemplos, cotações ou até mesmo algum caso especifico onde a palavra negativa não é dita em um sentido negativo. Em casos como estes um dicionário como o deste trabalho apresentaria como alerta uma mensagem que na realidade não é um, e vice versa.

Alertas incorretos podem ser falsos negativos ou falsos positivos, e cada um deles possui seus próprios problemas e maneiras de serem tratados em cada trabalho. Um falso negativo se baseia em um resultado negativo, porém na realidade deveria ser positivo. No caso deste trabalho um FN poderia ser, exemplificando, uma mensagem ofensiva ou perigosa, mas que não possui palavras presentes no dicionario da aplicação, então não seria considerado um alerta. Já um falso positivo é um resultado positivo, porém que na verdade deveria ser negativo. Um exemplo de FP no contexto deste trabalho seria uma frase irônica, com palavras presentes no dicionário porém em que a frase como um todo não tenha intuito negativo.

Em nosso trabalho, atualmente falsos positivos (FP) e falsos negativos (FN) são guardados em um mesmo banco e não há tratamento. Porém, uma mensagem FP pode ser marcada como "não alerta" pelo usuário manualmente. Apesar disso, para o futuro de nosso trabalho pretendemos incluir novas formas de classificação de mensagens, sendo uma delas uma inteligência artificial (IA). Com a inclusão de uma IA, atualizaríamos o banco para comportar Falsos positivos e negativos, e a inteligencia artificial terá acesso ao banco, de modo que o trabalho também vire uma forma de treinamento para ela.

3.3.2 APIs

Este trabalho atualmente só monitora usuários do Twitter com suas mensagens, utilizando a API do Twitter. Porém, a arquitetura foi feita de modo a estar aberta

para a inclusão do monitoramento de novas redes sociais futuramente.

Em um primeiro momento a decisão de ter o Twitter como primeira rede social a ser utilizada se deu pela grande quantidade de jovens usando a rede social. De acordo com nossos estudos envolvendo a API do Twitter para uso da mesma no projeto, descobrimos que ela é de fácil uso, porém como diversas outras APIs também possui suas limitações.

A principal limitação encontrada foi que a API do Twitter, através do uso gratuito, só oferece duas opções para recuperação de *tweets*.

Uma opção para consumo dos *tweets* ocorre através da busca por *tweets* do usuário. Nesse caso, pegaríamos uma grande quantidade de *tweets*, porém não recuperaríamos respostas e menções. Já a segunda opção seria pegar todos os *tweets* relacionados ao usuário monitorado em questão, porém só dos últimos 7 dias.

Entre outras redes sociais que gostaríamos de utilizar no futuro estão as do Instagram e LinkedIn. Entre estas duas, a API do LinkedIn apresenta uma perspectiva mais fácil de uso, pois há uma semelhança maior com a API do Twitter. Um grande exemplo são suas requisições e ambas usarem OAuth2.0, que valida a requisição por um Bearer Token. Porém, apesar de exigir mais modificações para consumir APIs que não usem o mesmo padrão, a arquitetura do sistema é expansiva o suficiente para aceitar o uso de outras APIs sem grandes problemas.

3.3.3 Requisitos de Sistema

Os requisitos de usuário são a apresentação e o esclarecimento conciso das funções e comportamentos necessários e esperados do sistema. Eles são identificados através de um processo chamado de elicitação de requisitos. O processo de elicitação deste projeto se baseou na escrita de uma completa descrição do sistema, e retirada dos requisitos a partir daí, e assim são encontrados os requisitos funcionais e não funcionais.

Dada a simplicidade e tamanho do projeto em questão, só foram identificados requisitos funcionais, que são as funcionalidades do sistema. Na Tabela 3.1 são

mostrados os requisitos funcionais do sistema.

Número	Descrição do Requisito
RF 01	O sistema permitirá que o usuário visualize um quadro de indivíduos monitorados, no qual aparece uma série de informações como nome de usuário, foto, último *tweet* e número de alertas.
RF 02	O sistema permitirá que o usuário visualize os *tweets* marcados como alertas de uma pessoa monitorada em uma extensão, ao selecionar o nome do monitorado.
RF 03	O sistema permitirá a visualização e edição de um perfil para o usuário, contendo informações como email usado para login, e mudança de senha.
RF 04	O sistema permitirá a visualização e edição de um dicionário base, que será o dicionário usado para a análise das mensagens resgatadas do perfil no Twitter do usuário monitorado.
RF 05	O sistema permitirá a criação e edição de um novo dicionário individual para cada pessoa sendo monitorada.
RF 06	O sistema permitirá a visualização de uma breve explicação sobre as funcionalidades do sistema, tanto para o usuário quanto para um visitante.
RF 07	O sistema permitirá que o usuário adicione uma pessoa monitorada ao informar um *username* e uma rede social
RF 08	O sistema deve permitir que o usuário pare de monitorar alguém a qualquer momento, ao selecionar a pessoa que deseja parar de monitorar.

Tabela 3.1: Requisitos de Sistema

3.3.4 Casos de Uso

Os casos de uso, diferentemente dos requisitos de usuário, não demonstram as funcionalidades do sistema, mas sim ações e interações que os vários tipos de usuários do sistema, chamados atores, tem com o sistema. A Tabela 3.2 demonstra os atores do sistema, ou seja, tipos de usuário. Já a Tabela 3.3 mostra uma descrição informal dos casos de uso.

Ator	Descrição do Ator
Visitante	Usuário antes de cadastrar-se no sistema.
Usuário	Como é chamado o visitante após seu cadastro e login no sistema.

Tabela 3.2: Atores do Sistema

Nome:	Cadastrar-se
Ator:	Visitante
Descrição:	Um visitante pode cadastrar-se no sistema e se tornar um usuário.
Nome:	Realizar Login
Ator:	Visitante
Descrição:	O usuário pode fazer login no sistema, se o mesmo já tiver realizado o cadastro.
Nome:	Visualizar o Dashboard
Ator:	Usuário
Descrição:	Visualização do dashboard com monitorados.
Nome:	Visualizar Alertas
Ator:	Usuário
Descrição:	O ator pode visualizar alertas de mensagens monitoradas.
Nome:	Visualizar perfil de usuário
Ator:	Usuário
Descrição:	O ator pode visualizar o perfil de usuário.
Nome:	Visualizar dicionário base
Ator:	Usuário
Descrição:	O ator pode visualizar dicionário base.
Nome:	Editar dicionários
Ator:	Usuário
Descrição:	O ator pode editar o dicionário base e os pessoais, acrescentando e retirando palavras.
Nome:	Editar perfil de usuário
Ator:	Usuário
Descrição:	O ator pode ditar perfil de usuário, podendo mudar username, senha e email.
Nome:	Adicionar monitorados
Ator:	Usuário
Descrição:	O ator pode adicionar um usuário do Twitter ao dashboard de monitoramento.
Nome:	Visualizar página inicial
Ator:	Usuário
Descrição:	Visualizar página inicial do sistema.
Nome:	Parar um monitoramento
Ator:	Usuário
Descrição:	o ator pode parar de monitorar um usuário do twitter.
Nome:	Deslogar do sistema
Ator:	Usuário
Descrição:	O ator pode realizar *Sign out* do sistema.

Tabela 3.3: Tabela de Casos de Uso

Outra forma de se representar os casos de uso é por um diagrama de casos de uso. Esse diagrama demonstra de forma visual os relacionamentos entre os atores e

seus casos de uso, como mostrado na figura 3.2.

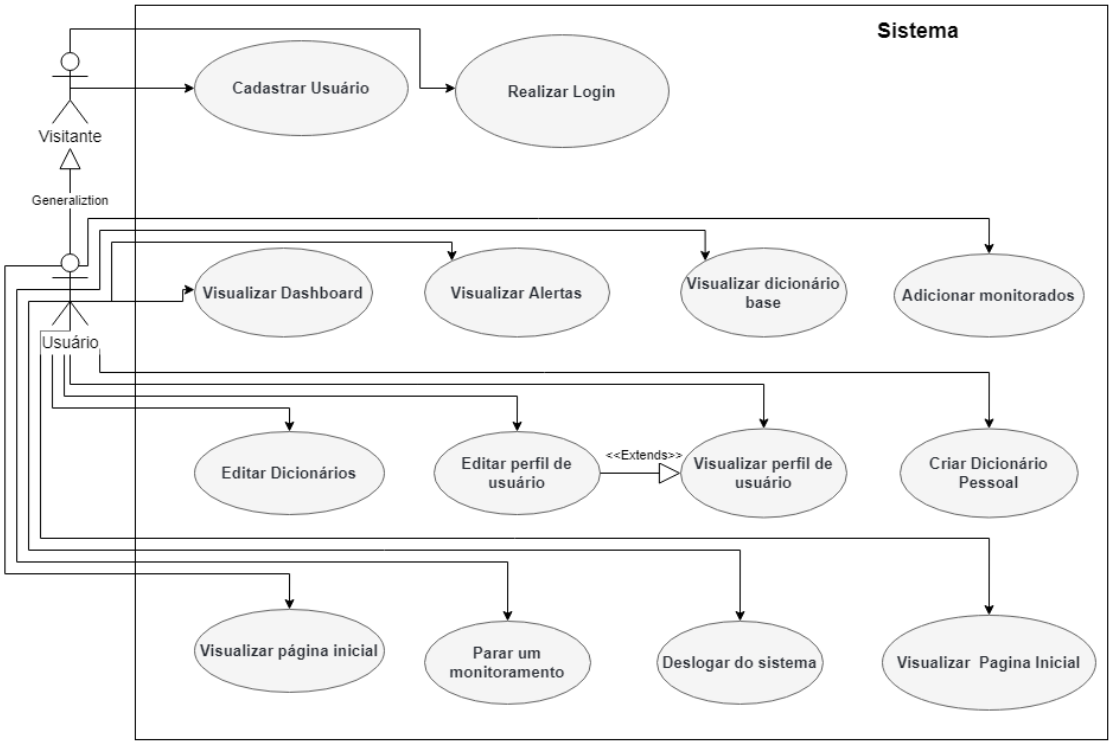

Figura 3.2: Diagrama de casos de uso do sistema

Capítulo 4

Implementação

Neste capítulo serão apresentadas a descrição e arquitetura do sistema. Além disso, também discorreremos sobre as tecnologias utilizadas em sua implementação em geral.

4.1 Tecnologias Utilizadas

Uma das maiores dificuldades no desenvolvimento *web* é a quantidade de tecnologias a serem aprendidas para a construção da aplicação. Esse conjunto de tecnologias necessárias para o desenvolvimento e funcionamento de uma aplicação é chamado de *Tech Stack* e, no caso da *web*, é composto por um sistema operacional, um servidor *web*, um banco de dados e ao menos uma linguagem de programação.

Quando os sistemas *web* se tornaram muito complexos os primeiros *frameworks* foram criados. Dessa forma, um *Tech Stack* mais tradicional das aplicações *web* requer conhecimento de JavaScript, CSS e HTML para o desenvolvimento da parte visual, também chamada de *front-end*, e conhecimento de um *framework back-end* como, por exemplo, o Spring, que é baseado na linguagem JAVA.

No entanto, à medida que os *frameworks* foram recebendo atualizações, seus usos tornaram-se muito complexos, perdendo assim seu papel de facilitadores no processo de desenvolvimento e originando uma nova dificuldade. Isso porque eles passaram a

ter uma curva de aprendizado muito grande e consumir tempo com configurações.

Dessa forma, quando o *framework* Ruby on Rails, também conhecido como Rails, foi lançado revolucionou o desenvolvimento *web*. Este *framework* utiliza a ideia de padronização antes de configuração, ou seja, a não ser que o desenvolvedor deseje modificar a configuração padrão do Rails, não há consumo de tempo com configurações do *framework* para o desenvolvimento de aplicações (RUBY; COPELAND; THOMAS, 2020).

A partir do Rails vários outros *frameworks* que seguiam a mesma filosofia foram lançados. Dentre eles alguns poucos são parte de um movimento chamado *Full Stack JavaScript*, que consiste em utilizar a linguagem JavaScript tanto no *front-end* quanto no *back-end*. Assim, é necessário o conhecimento de apenas uma linguagem de programação para o desenvolvimento de uma aplicação *web*.

Existem poucos *frameworks* que seguem a filosofia Rails para JavaScript. Um dos que surgiu desse movimento é o AdonisJS. Ele é um *framework* para Node.js que resulta em aplicações escritas em Typescript, uma linguagem fortemente tipada derivada do JavaScript. Ele possui um ORM (*Object Relational Mapper*) para vários bancos de dadose um deles é o PostGres[1] que foi escolhido para ser usado nesse trabalho pela sua robustez e por ser *open-source*, diferentemente do MySQl.

Nas próximas subseções serão discutidas mais a fundo as características de cada tecnologia utilizada, seus prós e contras e os motivos para sua escolha.

4.1.1 Node.js

O Node.js consiste em um software de código aberto que permite a execução de códigos JavaScript no servidor, e dessa forma, a execução do código não depende do navegador do cliente.

Dentre as principais características do Node.js que o fazem ser uma ótima opção destacam-se sua grande capacidade de escalabilidade, sua leveza e cache rápida, sua aplicabilidade para desenvolvimento de aplicações para diversas plataformas, e por

[1] https://www.postgresql.org/

último sua facilidade, devido tanto a utilizar só uma linguagem e possuir extenso suporte da comunidade, quanto à presença do *Node Package Manager* (NPM). Este último tem a função de permitir o uso de diversos pacotes de código aberto e reusável, o que diminui o tempo de desenvolvimento e consequentemente aumenta a qualidade do código.

Além disso, dentre as vantagens de se usar o Node também está a sua ótima performance em multitarefas, graças à sua grande capacidade de realizar múltiplos *requests* sem congestionar a memória RAM. Além disso, também possui IO/Não bloqueante, que é explicado brevemente a seguir.

Operações de entrada e saída também chamadas de operações E/S ou operações I/O, são operações envolvendo todos os outros dispositivos ou peças de um computador que não são a memória e o processador. Além disso, também podem ser operações de leitura e processamento de dados, *input e output*, e também acesso a algum servidor, a arquivos ou ao banco de dados. Estas operações muitas vezes envolvem um processo pausar outro até ser concluído, ou seja, são processos E/S bloqueantes: processos de determinado tipo, que quando começam, bloqueiam a resolução de outros tipos de processo até ele finalizar. Isto pode se tornar um problema quando se temos que resolver muitas operações diferentes.

O NodeJS apresenta a opção de se usar I/O não bloqueante, ou seja, processos assíncronos que não necessitam esperar pela finalização de outros processos para iniciarem. Vale ressaltar que ainda podemos usar métodos síncronos, já que estes se fazem necessários em alguns casos, apesar de o uso de processos assíncronos trazer uma grande rapidez, pois não há grande parte da espera causada por um processo bloqueado.

4.1.2 AdonisJS

O AdonisJS é um *framework* para Node.Js que possui uma grande gama de recursos acessíveis desde sua instalação, e facilidade de integração com diversos outros recursos, o que torna possível a criação de uma aplicação *web* apenas com ele. Além disso, esse *framework* utiliza a arquitetura MVC(Arquitetura Model-View-

Controller), que faz com que as classes sejam desacopladas e coesas, resultando em um código de fácil manutenção e um sistema escalonável.

Este sistema foi construído usando o AdonisJS, pois por padrão com ele se é utilizada a Linguagem Typescript, além de o framework também ter sido criado com essa mesma linguagem. Algumas vantagens dessa linguagem são que, além de possuir todos os recursos do Javascript, ela também fornece verificação de tipo estático e permite a criação de objetos baseados em classe, suportando assim herança e encapsulamento como outras linguagens de programação orientadas a objeto. Segundo Rana e Virk (2021), o próprio AdonisJS é escrito em Typescript.

A fim de seguir a ideia de *Full Stack JavaScript* a *template engine Edge* foi criada para uso em conjunto com o AdonisJS, porém também pode ser usada sozinha. Essa *engine* está sendo usada para auxílio na criação da parte visual deste sistema, pois pode renderizar HTML, *Markdown* e arquivos de texto. Ela também permite o uso de qualquer expressão Javascript. Ademais, a curva de aprendizado é pequena pois a escrita se assemelha muito à da linguagem Javascript (RANA; VIRK, 2021), auxiliando na simplificação e diminuição de tempo necessário na escrita e entendimento do código.

4.1.3 PostgreSQL

O AdonisJS oferece suporte para diversos bancos de dado SQL, que são facilmente integrados com o *framework*. Dentre eles, o escolhido para construção deste sistema é o PostgreSQL.

O *PostgreSQL* é um banco de dados *open source* e compatível com os principais sistemas operacionais, como Windows, Linux e MacOS. Ele também é conhecido e com uma grande comunidade de usuários, o que resulta em uma grande quantidade de material para consulta na Internet. Ademais, o PostgreSQL ainda recebe atualizações e dispõe de diversos recursos que permitem que o banco de dados do sistema evolua com o tempo.

4.1.4 TailwindCSS

O *TailwindCSS* é um *framework* para CSS que visa facilitar a estilização de sistemas. Esse *framework* disponibiliza inúmeras classes CSS prontas para uso direto no HTML, que estilizam características das páginas como cores, tamanho, sombreamento, dentre outros. Além disso, elas são responsivas e possuem nomes intuitivos para facilitar o uso. Neste trabalho, as classes desse *framework* foram empregadas nas marcações HTML construídas com o auxílio da Edge.

O TailwindCSS, por ser um *framework* de baixo nível, permite que a estilização seja feita do zero, conferindo assim um visual singular para cada aplicação que o utiliza. Porém, também é disponibilizado o TailwindUI, que é uma coleção de componentes prontos, como tabelas e barras de navegação, a fim de servir de base para desenvolvedores que, em lugar de criar um componente do zero, podem estilizar um já existente. Além disso, o pacote final do TailwindCSS tenta gastar uma quantidade mínima de memória pois, ao colocar o sistema em produção, ele exclui todas as classes CSS não utilizadas.

4.1.5 Bibliotecas de Destaque

O NodeJs possui diversas bibliotecas para auxílio na codificação de sistemas *web*. Dentre elas, há duas bibliotecas de extrema importância para este projeto: Adonis-scheduler[2] e a Natural[3].

O Adonis-scheduler, também chamado de Adonis Scheduler Provider, é uma biblioteca disponível por NPM, feita com o intuito de programar tarefas recorrentes em projetos Adonis.JS de maneira fácil e rápida. Ela foi utilizada e se tornou essencial neste projeto para fazer o cronjob, que faz a tarefa de recuperar novos tweets em um determinado período de tempo de a cada 15 minutos. A implementação foi simples graças à biblioteca, bastando a instalação e configuração da mesma.

Natural é um pacote facilitador de processamento de linguagem natural para

[2]https://github.com/nrempel/adonis-scheduler
[3]https://www.npmjs.com/package/natural

Node.Js. Ele apresenta diversas funcionalidades que permitem auxiliar na análise e classificação de palavras, e uma dessas funcionalidades que é a principal utilizada neste projeto é a *bag of words*. Ela permite a criação de um dicionário, consistindo de palavras julgadas de cunho negativo, e que pode também ser editado adicionando ou removendo palavras. Frases recuperadas são então analisadas e palavras presentes no dicionário são então identificadas nessas frases. Isto permite a análise dos tuítes de maneira mais simplificada.

4.1.6 API do Twitter

Para desenvolver com a API do Twitter é necessário primeiramente uma conta de desenvolvedor, sendo este em si um processo simples, somente exigindo informações necessárias, possíveis vínculos estudantis e razão para o uso da API. Somente com esse processo podemos criar e modificar projetos na plataforma de desenvolvedor do Twitter, e exclusivamente através desses projetos teremos acesso às credenciais necessárias para acessar os *endpoints* da API.

Essas credenciais, chaves e *tokens*, precisam ser utilizadas a cada *request* da API. Podem ser caracterizados como chaves e *secrets* da API, Bearer tokens ou *user access tokens*, cada um com uma função específica.

Chaves e Secrets são o que dá acesso ao projeto criado, e são gerados em sua criação. Permitem gerar os outros *tokens* necessários para as chamadas à API como o *User access token* e o *app access token*. Destes, o *app access token*, também chamado de *Bearer Token* é o utilizado pelas chamadas de nossa aplicação e sua diferença para com o *token* anterior, é que ele faz chamadas para a API em nome de uma aplicação específica, ao invés de em nome do usuário da aplicação, o que permite que qualquer pessoa utilizando a aplicação possa fazer requisições sem necessitar de demais permissões. Podemos utilizá-lo já que nossa aplicação só necessita de permissões de leitura, e nenhuma de escrita.

Uma parte de extrema importância na utilização desta API foi o uso das bibliotecas twitter-text, que é uma coleção de bibliotecas fornecidas em conjunto com a API do Twitter, e possui como seu principal objetivo disponibilizar maneiras de tokenizar e

realizar análises sintáticas de textos do Twitter. Ambas as ações são essenciais para a recuperação e tratamento de textos do Twitter.

4.2 Arquitetura do sistema

Aqui falaremos sobre a arquitetura MVC utilizada em nosso sistema, suas camadas e os detalhes de sua aplicação no sistema.

4.2.1 Arquitetura Model View Controller (MVC)

O modelo arquitetural Model View Controller, também conhecido como MVC, em sua versão clássica divide a aplicação em três camadas: *Models* ou Modelos, *Views* ou Visualizações e *controllers* ou Controladores. Algumas variações desse modelo, como a utilizada neste projeto, incluem também a camada *Services* ou Serviços.

A camada *Models*, segundo Pop e Altar (2014), é a parte do sistema que gerencia todas as tarefas relacionadas aos dados: validação, estado de sessão e controle, estrutura de fonte de dados (banco de dados). Ela também encapsula métodos para acessar os dados (bancos de dados, arquivos, etc.) e se torna uma biblioteca de classe reutilizável (POP; ALTAR, 2014). Esses métodos são responsáveis por operações de criação, leitura, atualização e deleção (CRUD) dos dados e são utilizados pela camada do *Controller* ou do *Service*, dependendo da variação do modelo MVC adotado pelo sistema.

A camada *Views*, de acordo com Pop e Altar (2014), é responsável pelo gerenciamento gráfico da interface do usuário. Isso significa dizer que todos os formulários, botões, elementos gráficos e todos os outros elementos HTML que estão dentro da aplicação. Essa camada recebe dados dos usuários do sistema e envia os eventos para a camada *Controller* e sua atualização depende da atualização dos dados que ocorre na camada *Models*.

A camada *Controllers*, como afirma Pop e Altar (2014), é responsável por manusear eventos. Esses eventos podem ser acionados por uma interação do usuário

com o sistema ou por um processo do próprio sistema. Ao receber um requisição, o *Controller* se comunica com o *Model* ou com o *Service* para recuperar os dados necessários para gerar a *View* necessária.

A camada *Services* contém toda a regra de negócio do sistema, por isso é nela que ocorre a validação dos dados de acordo com essas regras. O *Service* recebe os dados a serem modificados do *Controller* e, após validá-los de acordo com as regras de negócio do sistema, se comunica com o *Model*, enviando os dados que precisam ser manuseados por ele.

Como citado anteriormente, neste sistema foram implementadas as quatro camadas: *Models*, *Views*, *Services* e *Controllers*. A seguir discorreremos sobre os detalhes da implementação de cada camada.

4.2.2 *Models*

Na implementação dos *Models* utilizamos a abordagem de *Thin Models* que, como afirma Pop e Altar (2014), diz que o *Model* deve ser o mais simples possível, apenas abrangendo processamento de dados que são estritamente ligados ao objeto real que está sendo modelado. O autor diz também que uma vantagem dessa abordagem é que o *Model* se torna altamente reutilizável entre aplicações.

Dessa forma, neste sistema os *Models* compreendem apenas os dados das tabelas existentes no banco de dados, seus relacionamentos e métodos de processamento desses dados que são nativos do próprio *framework* Adonis.js. Três *Models* foram criados neste sistema: *Users*, *Monitoreds* e *Messages*.

O *Model Users* representa os usuários do sistema e contém informações como, por exemplo, e-mail, nome e senha do usuário. O *Model Monitoreds* representa as pessoas monitoradas por usuários do sistema, guardando informações como rede social e *username* dos monitorados. Por fim, o *Model Messages* representa as mensagens das pessoas monitoradas e retém dados como conteúdo e autor da mensagem.

Messages e *Messages* se relacionam de forma um para muito (*has many*), isto é, uma pessoa monitorada pode ter diversas mensagens, porém uma mensagem pode

pertencer apenas a um monitorado. Já o relacionamento entre *Monitoreds* e *Users* é de muitos para muitos (*many to many*), isso porque um usuário pode monitorar zero ou mais pessoas, da mesma forma que uma pessoa pode ser monitorada por um ou mais usuários.

Os *Models Users* e *Messages* também possuem um relacionamento de muitos para muitos. Isso se deve ao fato de que não só um usuário pode ter acesso a várias mensagens, como também, uma mensagem pode ser acessada por diversos usuários que estejam, por exemplo, monitorando a mesma pessoa.

Os relacionamentos *many to many* precisam de uma tabela *pivot* para armazenar os dados em comum aos dois *Models* relacionados. Por isso, para o funcionamento deste sistema foram criadas cinco tabelas: as tabelas users, monitoreds, messages, users_monitoreds e users_messages, sendo as duas últimas tabelas pivot para os relacionamentos dos *Models Users* e *Monitoreds* e dos *Models Users* e *Messages* respectivamente.

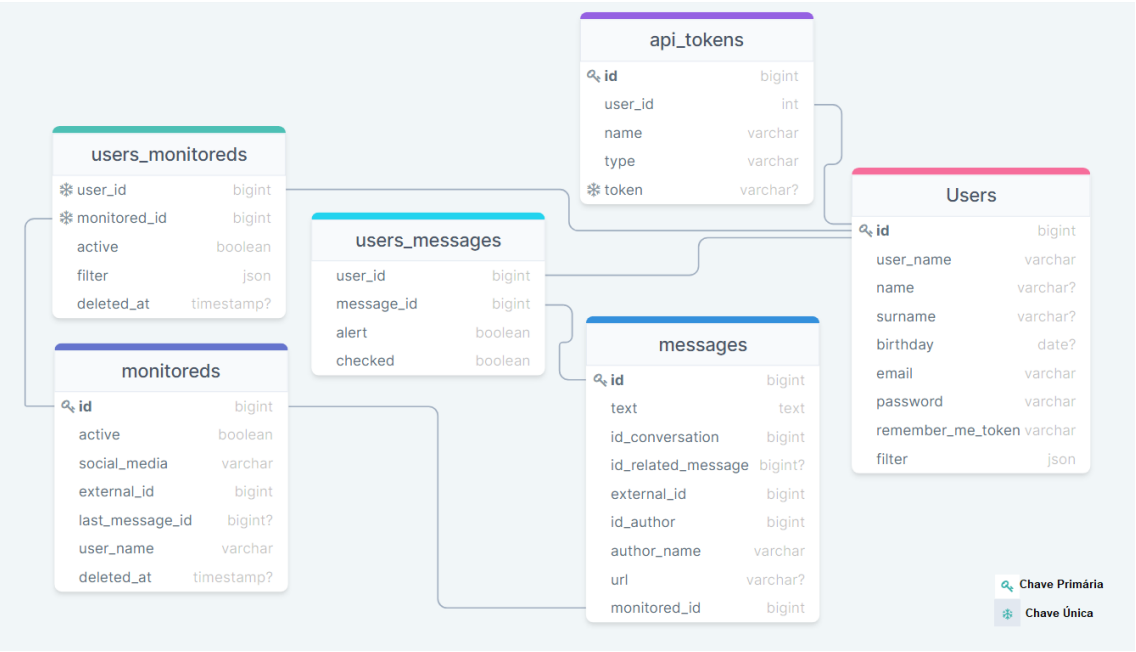

Figura 4.1: Tabelas do Sistema

4.2.3 *Services*

A camada *Services* deste sistema é formada por duas classes: MonitoredsService e TwitterService. Considerando a baixa complexidade do sistema, optamos por construir essas classes com métodos estáticos que funcionam como *helpers*, que contém validações e formatações de dados de acordo com as regras de negócio e podem ser chamados pelos *Controllers* para servir como uma ponte de comunicação entre eles e os *Models*, ou entre outros *Services* que precisem de métodos que já foram implementados.

A classe *MonitoredsService* contém métodos que lidam com os dados das pessoas monitoradas por usuários do sistema e suas mensagens. Um exemplo desses métodos é o *saveMonitoreds*, que recebe os dados de uma nova pessoa a ser monitorada e do usuário tentando monitorá-la, valida esses dados de acordo com a regra de negócio pertinente e aciona o *Model Monitoreds* para processar os dados em caso de sucesso na validação ou retorna um erro caso a validação falhe.

A classe *TwitterService* contém métodos que fazem requisições para a API do Twitter. Um exemplo é o método *getTweets*, que recebe dados das pessoas que tem seu Twitter monitorado por usuários do sistema e faz uma requisição para a API do Twitter para recuperar os tweets mais recentes que foram escritos por ou para essa pessoa. Ao recuperar os tweets, os dados são formatados para seguir a estrutura do *Model Messages* e então são enviados para o método *storeMessages*, da classe *MonitoredsServices*, que valida os dados e aciona o *Model Messages* para salvar as mensagens.

Os métodos do *TwitterService* são, no geral, utilizados pelo *MonitoredsService*, com exceção do *getTweets* que, em especial, é acionado por um cronjob. O cronjob inicia esse método a cada quinze minutos, um intervalo de tempo que foi estipulado a fim de não ultrapassar o limite de chamadas da API do Twitter e ao mesmo tempo evitar a perda de tweets ao ficar muito tempo sem buscar por novas mensagens.

É importante ressaltar também que, ao ser acionado, o método *getTweets* busca por novos tweets de todos os monitorados da rede social Twitter, fazendo uma

requisição para cada monitorado. Isso pode causar alguns problemas, um deles é a captação de mensagens repetidas, ou seja, que já foram recuperadas anteriormente e salvas no banco. Nós solucionamos esse problema ao salvar em toda busca o last_message_id, que é o identificador da mensagem mais recente que foi retornada na busca e retornar esse valor na próxima busca como o since_id, que indica a partir de que mensagem a busca deve retornar.

Um outro problema que pode ocorrer é um monitorado enviar mais tweets do que o número de tweets que retornam na consulta. A solução para esse problema é checar se a resposta vinda da possui o atributo next_token e, caso possua, realizar a consulta novamente, dessa vez passando o nex_token, até que a resposta não possua esse atributo.

4.2.4 *Controllers*

A camada *Controllers*, neste sistema, é constituída de duas classes. Uma delas, a *AuthController*, é responsável por lidar com eventos de autorização de usuários, como cadastrar, conectar e desconectar usuários. Já a *MonitoredsController* é responsável por eventos que envolvem atualização de dados do usuário como, por exemplo, deixar de monitorar uma pessoa ou atualizar o perfil.

Porém, antes de chegar ao *Controller* o evento precisa passar por um *Middleware* e uma *Route*, ou rota. Um *Middleware*, segundo Rana e Virk (2021), é uma série de funções executadas antes da requisição chegar à rota. No caso deste sistema, quando um usuário não conectado tenta acessar uma página permitida apenas para usuários conectados ele é redirecionado para a tela de *login*.

Ao passar pelo o *Middleware* o evento chega à rota, que é responsável por relacionar a requisição feita em uma URL específica a um método de um *controller*. Se um evento não mapeado ocorrer, um erro é retornado e o usuário é direcionado para uma tela de página não encontrada.

4.2.5 *Views*

As *Views* deste sistema, como citado anteriormente, foram desenvolvidas, principalmente, com o auxílio de três ferramentas: TailwindCSS, AlpineJS e *template engine Edge*. O AlpineJS foi utilizado para criar elementos dinâmicos nas páginas, enquanto o TailwindCSS funciona como estilizador das mesmas.

A *template engine Edge*, além de renderizar as páginas, nos permitiu atingir uma grande reusabilidade de código com seu sistema de componentes e *layouts*. Um exemplo disso são os formulários, todos derivados de um mesmo componente Edge. Essa reutilização de código proporciona também maior facilidade em manter um estilo consistente em todo o sistema, pois só é preciso estilizar um único componente e replicar o mesmo em todo o sistema.

Dentre as principais *Views* deste sistema estão as *Views* da página inicial demonstrada na Figura 4.2, e as de cadastro e login, mostradas nas Figuras 4.3 e 4.4 respectivamente, que garantem acesso a este sistema, e ao *dashboard* que permite ao usuário visualizar as pessoas que ele está monitorando. As telas de cadastro e login são constituídas, principalmente de um formulário, que é derivado de um mesmo componente Edge, e um *layout*, que engloba as barras de navegação e é difundido em todo o sistema.

Figura 4.2: Página Inicial do Sistema.

A página inicial mostra uma imagem de um olho estilizado olhando por um monóculo, demonstrando o olhar cuidadoso que pretendemos disponibilizar com o sistema, e que foi criada para o projeto. Além disso ela também apresenta os botões de login e cadastro. Já as páginas de *Sign Up* e *Login* também apresentam a mesma imagem, além do formulário com as informações necessárias para as ações.

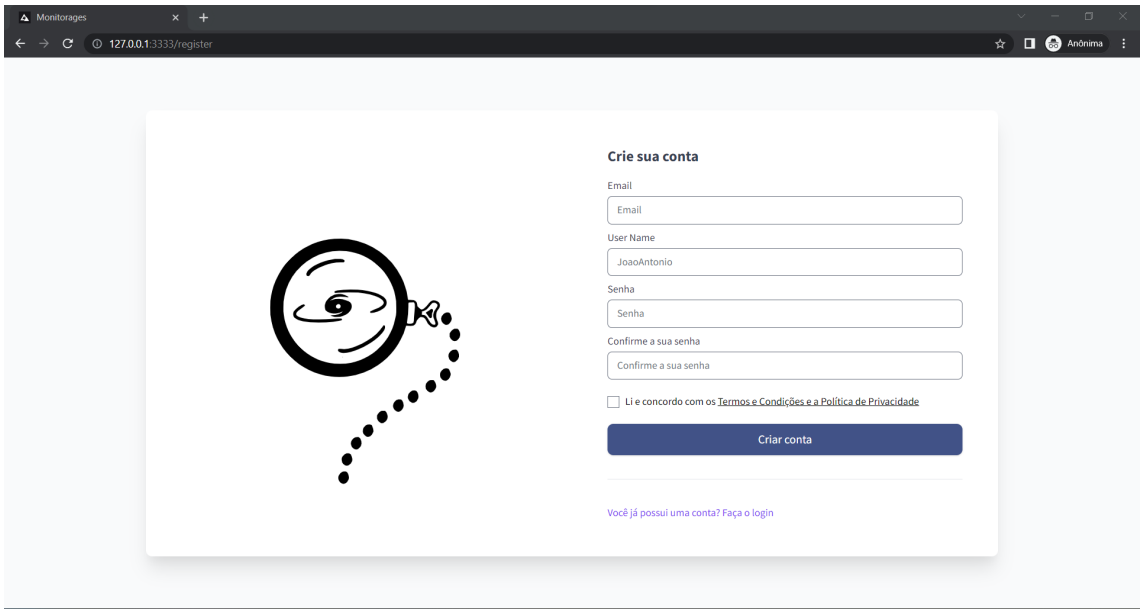

Figura 4.3: Página de Cadastro.

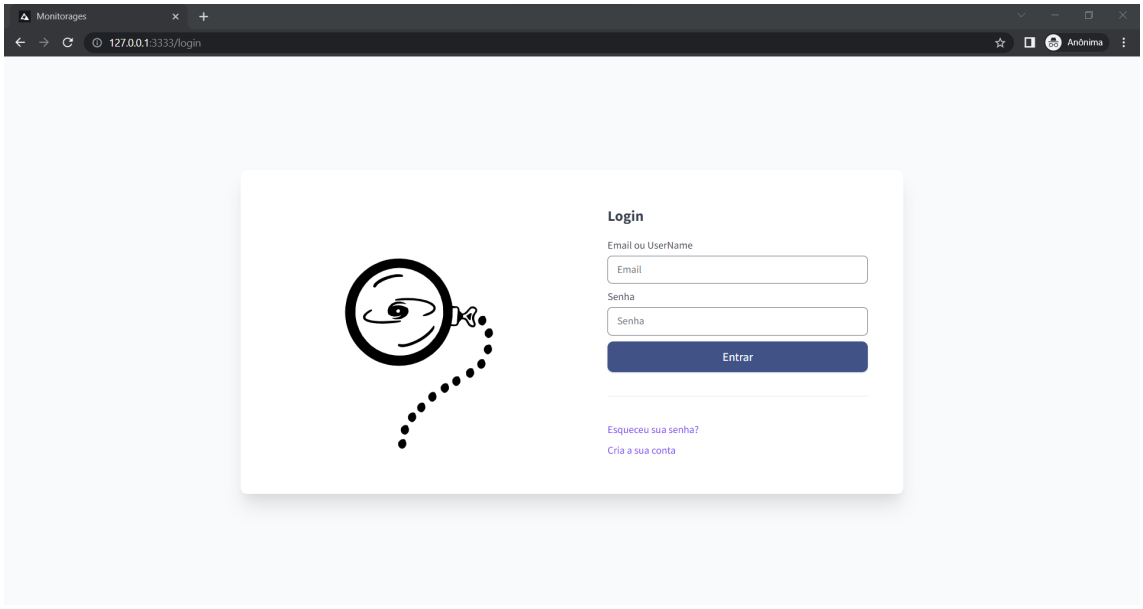

Figura 4.4: Página de Login.

Já o *dashboard* é constituído por uma tabela, que é preenchida por uma lista de pessoas que o usuário monitora. Cada pessoa monitorada é representada em uma linha da tabela e possui um botão de editar, que edita o dicionário de classificação da mensagem daquela pessoa monitorada, e um botão de excluir, que faz o usuário deixar de monitorar aquela pessoa como mostra a Figura 4.5.

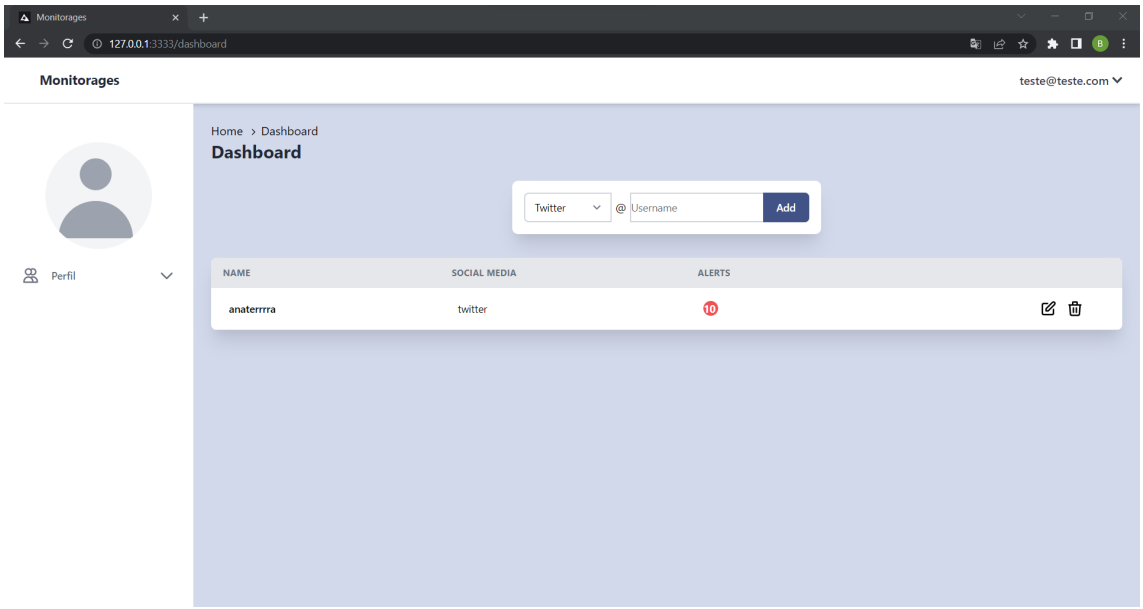

Figura 4.5: *Dashboard* mostrando somente usuários.

Além disso, ao clicar na linha da pessoa desejada, abre-se um *dropdown* que revela as últimas mensagens classificadas como uma ameaça pelo sistema e que o usuário ainda não marcou como visualizadas. Cada mensagem popula uma linha do *dropdown*, que também é uma tabela, e possui seu próprio botão de *check*. Ao apertar esse botão o usuário está marcando a mensagem como visualizada, fazendo assim com que ela não seja mais exibida como apresentado na Figura 4.6.

O *dashboard* também possui o *layout* que, como citado anteriormente, fornece à página os menus de navegação e um formulário para a adição de novas pessoas na lista de monitorados do usuário. Esse formulário permite que o usuário informe a rede social e o nome de usuário da pessoa que ele pretende monitorar. Caso o usuário já monitore aquela pessoa, ou o sistema não consiga encontrar a pessoa desejada, o formulário retorna uma mensagem que avisa o usuário do erro.

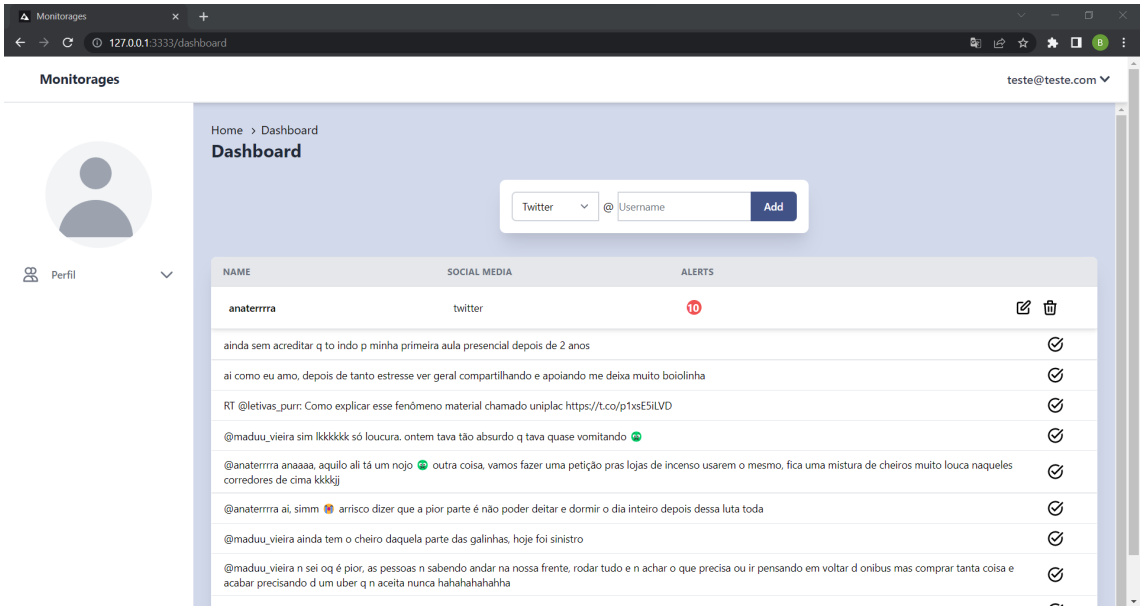

Figura 4.6: *Dashboard* mostrando as mensagens.

Capítulo 5

Conclusão

Este capítulo apresentará as ponderações finais sobre o trabalho, retomando a importância do sistema em questão, seus objetivos realizados e não realizados, ideias futuras para complementação e melhora desse trabalho, e pensamentos sobre resultado final do trabalho.

5.1 Considerações finais

Como já demonstrado nos capítulos anteriores, o número de jovens utilizando redes sociais como o Twitter é alto, e por muito tempo só vem crescendo. Além de o mais, os números de casos de *cyberbulling*, *phishing*, assédio, entre outros riscos, também aumenta para este grupo. Vale ressaltar além deste fato que grande parte dos tweets desses jovens não são privados, ou então são respostas para tweets abertos, portanto estes são acessíveis por todos os públicos. Este cenário, juntamente com a falta de tempo de pais e responsáveis, que muitas vezes trabalham fora e possuem outras obrigações, cria um ambiente onde o jovem não tenha alguém confiável que possa servir de apoio e conselheiro.

Sendo assim, em nossa visão, este trabalho apresenta uma ótima alternativa para o monitoramento de contas de jovens na rede social do twitter, já que engloba monitoramento independentemente da quantidade de twittes que o jovem pode criar

por dia, além de não invadir a privacidade pois só vê twittes não privados. Com pouco tempo do dia sendo usado, o usuário do sistema pode monitorar as contas e descobrir diversos alertas, além de analisá-los com somente um clique.

5.2 Limitações e trabalhos futuros

Em conclusão, diversos objetivos deste trabalho foram atingidos, inclusive o objetivo principal: a possibilidade de monitoramento de diversas contas do Twitter, de maneira fácil para o usuário do sistema. Porém, em contra partida, algumas funcionalidades pretendidas não foram possíveis de serem implementadas.

O trabalho apresenta uma boa escolha para um leve e conciso sistema para monitoramento de tweets. Porém não de outras plataformas. A maior limitação que encontramos foi a implementação de *requests* para recuperação de mensagens utilizando outras API além da do Twitter.

A implementação do banco de dados foi completada de maneira a permitir a inclusão de novas redes sociais sem necessitar mudanças no contexto de dados. Além disso, a interface também foi feita pensando em uma implementação extensiva pra outras redes sociais, da mesma maneira que a camada de dados. Porém, vale ressaltar que o projeto ainda assim precisaria de diversas mudanças em seu código para essa inclusão acontecer, pois necessitaria de mudanças e inclusões de código nas partes de *models* e *services* para cada nova rede social. Além disso, também necessitaríamos de APIs ou *crawlers* de outras redes sociais para conseguirmos gerar novos *requests*.

Além disso, um dos objetivos, a criação de gráficos mostrando informações sobre as quantidades de mensagens e de alertas não pode ser implementado a tempo da criação deste texto. Também pudemos implementar a presença de um perfil de usuário e edição deste perfil, além de a possibilidade de manter o usuário com login ativo no sistema até o mesmo fazer sign out.

Pudemos implementar um dicionário para a classificação das mensagens, porém não foi possível, devido a falta de tempo, implementar outros meios de classificação de mensagens. Apesar disso, o sistema foi criado de modo a aceitar a implementação

de outros meios de classificações de mensagens. Inclusive, em nossos planos futuros está a inclusão de uma IA, que pode classificar e aprender com as mensagens do sistema, já que o usuário confirma se é ou não alerta, e melhorando assim a qualidade da classificação cada vez mais.

Vale também ressaltar que, apesar de nosso sistema até o momento só funcionar em navegadores e só classificar mensagens em português, temos os planos de criar uma versão para celulares em formato de aplicativo disponibillizado por download, e também de incluir diversas outras línguas, como por exemplo inglês, para assim o aplicativo estar disponível para cada vez mais pessoas ao redor do mundo.

Referências

ANG, R. P. *Adolescent cyberbullying: A review of characteristics, prevention and intervention strategies*. Elsevier Ltd, 2015. 35-42 p. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S1359178915000968?via%3Dihub>.

BERNERS-LEE, T. *Weaving the Web: The original design and ultimate destiny of the World Wide Web, by its inventor*. 1. ed. HarperCollins Publishers Inc., 10 East 53rd Street, New York, NY 10022.: HarperCollins, 2000. ISBN 0-06-251586-1.

CERN. *worldwideweb.cern*. 1990. <https://worldwideweb.cern.ch/browser/>. Acessado em 26/11/2021.

CHAN, T. K.; CHEUNG, C. M.; LEE, Z. W. Cyberbullying on social networking sites: A literature review and future research directions. *Information and Management*, Elsevier B.V., v. 58, 3 2021. ISSN 03787206.

FLANAGAN, D. *JavaScript - O guia Definitivo*. [S.l.: s.n.], 2011. ISBN 9788565837194.

GHOSH, A. K. et al. A matter of control or safety? examining parental use of technical monitoring apps on teens' mobile devices. In: . [S.l.]: Association for Computing Machinery, 2018. v. 2018-April. ISBN 9781450356206.

HALL, E. A. *Internet Core Protocols The Definitive Guide*. 1. ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly, 2000. ISBN 1-56592-572-6.

INFORMATICS, M. B. F. of et al. *Social Network Monitoring Application for Parents with Children Under Thirteen*. [s.n.], 2015. ISBN 9781479960491. Disponível em: <https://ieeexplore.ieee.org/document/7051456>.

KAHIMISE, J.; SHAVA, D. F. B. *An analysis of children's online activities and behaviours that expose them to cybercrimes*. 2019. ed. 27th Telecommunications forum TELFOR, 2019. ISBN 9781728147901. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8971089>.

KEMP, S.; HOOTSUITE; SOCIAL, W. A. *6 IN 10 PEOPLE AROUND THE WORLD NOW USE THE INTERNET*. 2021. <https://datareportal.com/reports/6-in-10-people-around-the-world-now-use-the-internet>. Acessado em 24/10/2021.

KEMP, S.; HOOTSUITE; SOCIAL, W. A. *DIGITAL 2021 OCTOBER GLOBAL STATSHOT REPORT*. 2021. <https://datareportal.com/reports/digital-2021-october-global-statshot>. Acessado em 24/10/2021.

MENINI, S. et al. *A System to Monitor Cyberbullying based on Message Classification and Social Network Analysis*. 2019. 105-110 p. Disponível em: <https://aclanthology.org/W19-3511.pdf>.

NETNANNY. *NetNanny*. 2018. <https://www.netnanny.com/>. Acessado em 20/10/2021.

PALOQUE-BERGèS, C.; SCHAFER, V. Arpanet (1969–2019). *Internet Histories*, Routledge, v. 3, p. 1–14, 1 2019. ISSN 24701483. Disponível em: <https://www.tandfonline.com/doi/abs/10.1080/24701475.2018.1560921>.

POP, D.-P.; ALTAR, A. Designing an mvc model for rapid web application development. *Procedia Engineering*, v. 69, p. 1172–1179, 2014. ISSN 1877-7058. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013. Disponível em: <https://www.sciencedirect.com/science/article/pii/S187770581400352X>.

POWERS, D. *Beggining CSS3 Mastering the Language of Web Design*. New York, 233 Spring Street, 6th Floor, New York, NY 10013: Apress, 2012. ISBN 978-1-4302-4473-8.

RANA, A.; VIRK, H. *A fully featured web framework for Node.js*. 2021. <https://docs.adonisjs.com/guides/introduction>. Acessado em 17/11/2021.

RUBY, S.; COPELAND, D. B.; THOMAS, D. *Agile Web Development with Rails 6*. 1. ed. Raleigh, North Carolina: The Pragmatic Bookshelf, 2020. ISBN 9781680506709.

TIRUMALA, S. S.; SARRAFZADEH, A.; PANG, P. A survey on internet usage and cybersecurity awareness in students. In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. [s.n.], 2016. p. 223–228. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7906931>.

VANDERHOVEN, E. et al. Involving parents in school programs about safety on social network sites. *Procedia - Social and Behavioral Sciences*, Elsevier BV, v. 112, p. 428–436, 2 2014. ISSN 18770428. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877042814012026>.

WINDRUM, P. Leveraging technological externalities in complex technologies: Microsoft's exploitation of standards in the browser wars. *Research Policy*, v. 33, p. 385–394, 4 2004. ISSN 00487333.