Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Veredito: uma plataforma de debates *online*

Rodrigo Orem da Silva

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisora: Profª. Drª. Kelly Rosa Braghetto

São Paulo
2023

Agradecimentos

Este trabalho é fruto de um esforço coletivo, que vai muito além das horas dedicadas ao desenvolvimento do sistema, à pesquisa e à escrita. Dentre as pessoas e instituições que tornaram esse trabalho possível, gostaria de agradecer:

- à minha família, pelo amor incondicional;
- à Profª. Kelly, por toda a assistência e conhecimento oferecidos durante este trabalho;
- à Universidade de São Paulo, pelo comprometimento com a produção de conhecimento científico;
- aos amigos que fiz durante a graduação, que tornaram os dias mais leves; em especial ao Evandro Nakayama, que será sempre lembrado pela sua generosidade, ambição e curiosidade.

Resumo

Rodrigo Orem da Silva. **Veredito: uma plataforma de debates *online***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Este trabalho desenvolveu o **Veredito**, uma plataforma de debates. Apesar da *web* ter se tornado central para o debate de temas de interesse público, existem desafios associados a discussões *online*, como ataques pessoais e falta de estrutura do conteúdo. Propomos um sistema *web* para abordar esses problemas, onde o debate organiza-se entre argumentos prós e contras. Finalmente, desenvolvemos um modelo classificador de tópicos para facilitar a organização do conteúdo.

Palavras-chave: Debate. Classificação de texto. Sistemas *web*.

Abstract

Rodrigo Orem da Silva. **Veredito: an online debates platform**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

This work developed **Veredito**, a debate platform. Despite the web having become central for discussing topics of public interest, there are challenges associated with online discussions, such as personal attacks and lack of content structure. We propose a web system to address these issues, where the debate is organized between pro and con arguments. Finally, we developed a topic classifier model to facilitate content organization.

Keywords: Debate. Text classification. Web systems.

Sumário

Introdução

Plataformas sociais são meios de comunicação importantes para o debate político e o exercício da cidadania em uma democracia (Ehrenfeld e Barton, 2019). O público que produz e consome as informações nessas plataformas facilita a criação de uma esfera pública crítica e racional que, teoricamente, poderia apoiar o debate público, a deliberação e a formação de comunidades, além de melhorar a participação dos cidadãos na tomada de decisões políticas (Snellen *et al.*, 2012).

No entanto, essas plataformas possuem diversos problemas, como as "bolhas ideológicas", que reforçam crenças pessoais e excluem opiniões opostas (Mounk, 2018). Acredita-se que uma das motivações para a permanência das pessoas em tais bolhas seja que o debate público virtual é poluído com ofensas e mal comportamento, afastando as pessoas das ideias divergentes.

Problemas da esfera pública *online*

Apesar dos *sites* de discussão *online* serem importantes para a troca de informações, a liberdade e o anonimato dos criadores de conteúdo representam um desafio para governá-los (Cheng *et al.*, 2017). Esses sites frequentemente contêm conteúdo indesejado, como spam, postagens abusivas e fora do tópico (Jhaver *et al.*, 2019).

Sites de notícias se tornaram um desses lugares para as comunidades discutirem questões comuns, desencadeadas por artigos de notícias. Alguns trabalhos avaliaram a discussão do debate *online* sobre esses sites e, apesar dos esforços para manter a discussão construtiva e cordial, muitos entrevistados de uma pesquisa classificaram o discurso da seção de comentários como ofensivo. Na mesma pesquisa, relatou-se os seguintes problemas do ambiente de comentários: distorção, anonimato, ataques, *flaming*[1], propagação de desinformação e prejuízos à reputação do jornal (Diakopoulos e Naaman, 2011).

A pesquisa concluiu que a *trollagem*[2] decorre de fatores inatos e situacionais, então é importante desenvolver novas funcionalidades para moderar esse conteúdo. Por exemplo, sugere-se que inferir o humor por meio do comportamento de postagem recente e permitir que os usuários retirem comentários postados recentemente pode ter um impacto positivo na redução da quantidade de contribuições de baixa qualidade (Cheng *et al.*, 2017).

[1] Discussão hostil entre usuários na internet por meio de mensagens ofensivas.

[2] Comportamento na esfera digital que busca desestabilizar uma discussão e fazer provocações para enfurecer os outros usuários.

Ferramentas de moderação que possuem muitos falsos positivos também podem gerar problemas. Por exemplo, é importante distinguir discurso de ódio de noticiabilidade, porque falhas em abordar questões de contexto pode ter consequências graves, como a persistência de desinformação no Facebook ou WhatsApp que provavelmente contribuiu para a violência em Mianmar (Jhaver *et al.*, 2019). Outro problema da remoção injusta de conteúdo dos usuários é que isso pode exacerbar o comportamento antissocial (por exemplo, ser punido injustamente pode fazer com que o usuário tenha um comportamento pior) (Cheng *et al.*, 2017).

Lidar com essas questões com eficiência também requer práticas sociais e tecnológicas nos sistemas de regulação das plataformas. Automatizar a moderação não apenas facilita a escalabilidade, mas também permite consistência nas decisões de moderação. Entre as soluções propostas para melhorar a qualidade de debate público estão a facilitação do desenvolvimento e compartilhamento de ferramentas de regulação automatizadas, além de sistemas melhorados onde os humanos trabalham conjuntamente com sistemas automatizados ao invés de desenvolver sistemas totalmente automatizados (Jhaver *et al.*, 2019; Chaudoin *et al.*, 2017).

Soluções existentes

Atualmente, o debate nas redes sociais ocorre por meio de postagens, de repostagens (em que um usuário republica certo conteúdo com seu próprio comentário), e de seções de comentários. Geralmente, os comentários mais populares (que receberam mais curtidas ou mais respostas) são exibidos primeiro. Os sites de notícias e agregadores de *links* sociais também possuem seções de comentários que permitem que os leitores comentem e escrevam réplicas aos outros comentários, geralmente em uma estrutura de árvore. Já os fóruns de discussão costumam exibir as mensagens de maneira cronológica e para responder a uma mensagem específica, ela é reproduzida na resposta.

Apesar desses meios terem ampla participação pública, os debates possuem baixa qualidade, como mencionado na seção anterior.

Para abordar esses problemas, um trabalho de Chaudoin *et al.* (2017) propôs o Kialo[3], uma plataforma de debate estruturado que foi projetada para repensar o papel do debate em pesquisa, divulgação científica e ensino, principalmente na área de ciência política. Atualmente o Kialo possui quase 18 mil debates e mais de 700 mil argumentos, de modo a ser a principal ferramenta de argumentação colaborativa publicamente disponível.

Essa plataforma divide os argumentos de cada tema de discussão entre prós e contras, e o debate é exibido visualmente em uma estrutura de árvore, conforme a figura 1. Acreditamos que, embora uma visualização em árvore da discussão seja intuitiva, ela pode dificultar a compreensão já que o conteúdo fica fragmentado e são necessárias várias interações com a aplicação para ler um argumento com profundidade. Além disso, o contexto de uma discussão pode ser perdido ao navegar na árvore, de modo que subargumentos mais profundos podem não ser aplicáveis à discussão original.

[3] Disponível em https://www.kialo.com/

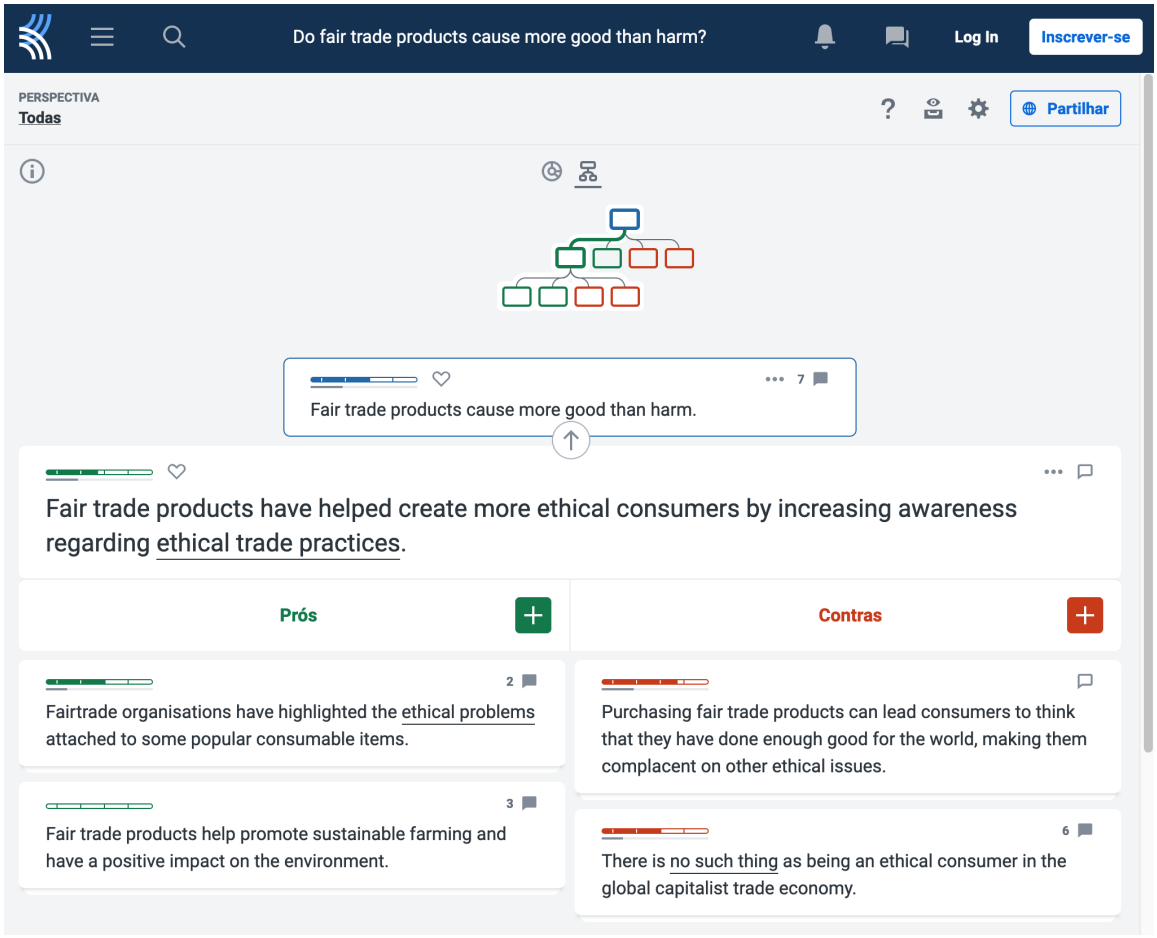

Figura 1: *Discussão (em inglês) no Kialo, estruturada em árvore*

Outro trabalho desenvolveu o Spirited[4], uma ferramenta de debate estruturado em que a discussão é feita ao redor de um artigo de opinião. Os argumentos são construídos na forma de anotações no artigo e os usuários podem votar nas anotações e nas sentenças do artigo. Os autores acreditam que manter uma estrutura menos rigorosa e mais livre favorece o engajamento e gera discussões mais interessantes (Mcelfresh, 2016).

No entanto, observamos que com essa estrutura de debates mais flexível, é mais difícil organizar, classificar e reusar partes da discussão. Além disso, plataformas de anotações colaborativas também enfrentam problemas de engajamento, de modo que o único caso de sucesso é o Genius[5], voltado à anotações em letras de músicas. O sucesso dessa plataforma pode ser atribuído a ela ter um propósito específico e um objetivo claro: o de interpretar o sentido das letras das músicas (Mcelfresh, 2016).

[4] Disponível em https://spirited.net/

[5] Disponível em https://genius.com/

Objetivos

Para abordar esses problemas, foi construído o **Veredito**[6], uma plataforma *web* de debates que pretende oferecer um ambiente mais propício para o debate público ao priorizar discussões de maior qualidade, estruturadas em argumentos prós e contras. Como se trata de uma plataforma aberta a colaboração pública, há desafios de moderação importantes para garantir que o conteúdo seja relevante aos usuários sem que isso afete a participação na plataforma.

Além disso, ferramentas de processamento de texto podem auxiliar o trabalho de moderadores e dos usuários ao ordenar e classificar o conteúdo publicado na plataforma. Neste trabalho, foi experimentado uma ferramenta de classificação de texto capaz de atribuir rótulos automaticamente a perguntas. Como é necessário um volume de dados significativo para treinar o classificador, técnicas de raspagem de dados foram utilizadas em uma plataforma de debates já existente.

[6] Veredito. https://veredito.org

Capítulo 1

Conceitos

Aplicações *web*, como o **Veredito**, são sistemas de *software* comumente compostos por dois componentes principais, o *frontend* e o *backend*. O *frontend* corresponde a uma interface gráfica, que é renderizada no navegador do usuário e se comunica com o *backend*, que roda em um servidor e é responsável pelo processamento dos dados. Geralmente, o *backend* também é responsável pela autenticação, comunicação com o banco de dados e validação das operações (Pavlenko *et al.*, 2020).

1.1 *Web*

A *World Wide Web*, ou apenas *web*, é um sistema de informação que permite o compartilhamento de conteúdo por meio da internet. A web foi idealizada para ser um repositório mundial de conhecimento humano, de modo a permitir o consumo das informações por qualquer pessoa, ligações entre tudo que pode ser visualizado e flexibilidade na maneira como as informações são recebidas e interpretadas (Berners-Lee *et al.*, 1994).

Um dos protocolos utilizados pela *web* para distribuir esse conteúdo é o protocolo HTTP (*HyperText Transfer Protocol*), que pertence a camada de aplicação do modelo OSI. O protocolo HTTP utiliza a arquitetura cliente-servidor, em que o servidor possui recursos que podem ser obtidos por um cliente, e esse cliente é o responsável por iniciar a conexão conforme sua necessidade (Pavlenko *et al.*, 2020). É por meio do protocolo HTTP que é feita a maior parte da comunicação entre o *backend* e o *frontend*.

1.2 Tecnologias do *backend*

Nesta seção, vamos descrever algumas das tecnologias usadas na construção do *backend* do **Veredito**.

1.2.1 Banco de dados

Como aplicações *web* costumam lidar constantemente com dados fornecidos ou solicitados pelos usuários, é comum o uso de sistemas gerenciadores de bancos de dados

(SGBDs). Existem diversos desafios para acomodar um número cada vez maior de usuários e de dados e uma variedade de solução de SGBDs para esses problemas.

Os SGBDs relacionais são os mais tradicionais e consistem em estruturar os dados com base no modelo relacional, que é consistente com a lógica de primeira ordem. Apesar desses sistemas oferecerem um misto de simplicidade, flexibilidade e desempenho, a dificuldade em torná-los escaláveis popularizou os SGBDs NoSQL na última década (Padhy *et al.*, 2011).

O SGBDs NoSQL não implementam completamente o modelo transacional ACID (sigla para os atributos atomicidade, consistência, isolamento e durabilidade). Essa falta de rigor em relação a um modelo de dados e a flexibilidade na normalização foram projetadas para permitir que esses SGBDs sejam mais facilmente distribuídos, de modo a aumentar a escalabilidade e o desempenho em alguns cenários (Győrödi *et al.*, 2020).

Para esse projeto, foi escolhido o PostgreSQL[1] como banco de dados da aplicação por ser muito completo em relação a funcionalidades, fácil de usar e com muito suporte na *web* devido a sua alta popularidade. Um SGBD baseado em grafos foi considerado porque representaria mais naturalmente o caso em que um argumento é utilizado em várias discussões, mas a ideia foi descartada por questões de usabilidade (abordado na seção 2.1).

1.2.2 Node.js

O Node.js[2] é uma plataforma de desenvolvimento que provê um ambiente de execução de programas escritos em JavaScript[3] no servidor. Como JavaScript é a linguagem usada para escrever aplicações no *frontend*, que rodam no navegador, o Node.js possibilita compartilhamento de código entre *backend* e *frontend*.

As principais características do Node.js são a arquitetura orientada a eventos capaz de E/S (entrada e saída) assíncrona. Isso permite que aplicações em Node.js sejam capazes de alcançar altas taxas de transferência e escalabilidade, o que torna a escolha dessa plataforma popular no contexto de aplicações *web*.

1.2.3 TypeScript

O JavaScript é uma linguagem de programação de alto nível predominante na *web*, que conta um sistema de tipos dinâmico. Linguagens fracamente tipadas são ideais para ter agilidade ao escrever pequenos programas, mas são inapropriadas para desenvolver e manter sistemas maiores (Bierman *et al.*, 2014).

Um sistema de tipos mais robusto é capaz de detectar erros no código antes que ele seja executado, pois um compilador é capaz de avaliar se a expressão aceita pelo sistema de tipos resulta em um valor do tipo apropriado. Uma linguagem fortemente tipada evita

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

[2] Node.js. https://nodejs.org/

[3] JavaScript. https://developer.mozilla.org/docs/Web/JavaScript

operações ilegais, oferece mais garantias quanto a segurança da memória e detecta erros lógicos relacionados a semântica de tipos.

O TypeScript[4] é uma linguagem de programação que adiciona tipos estáticos ao JavaScript. Como a sintaxe do TypeScript é um superconjunto da sintaxe do JavaScript, todos os programas em JavaScript são válidos em TypeScript, embora eles possam falhar na checagem de tipos. No entanto, uma das características de projeto da linguagem é permitir tipagem gradual para compatibilidade com bibliotecas que não são TypeScript. Como resultado, possíveis erros de tipos não identificados estaticamente podem permanecer ocultos até serem executados (Bierman *et al.*, 2014).

O TypeScript foi escolhido para a construção tanto da aplicação *frontend* quando *backend* por sua ampla disseminação, maior segurança e conforto no desenvolvimento em comparação ao JavaScript e possibilidade de compartilhamento de código entre os dois componentes.

1.2.4 *GraphQL*

O GraphQL[5] é um arcabouço conceitual para fornecer uma interface de acesso a dados na *web*. Esse arcabouço inclui um linguagem de consulta homônima. Essa interface de dados é uma alternativa ao padrão REST e tem a vantagem de fornecer ao cliente toda a informação desejada em um único pedido. Com REST, frequentemente é necessário fazer vários pedidos e os pedidos podem incluir informação desnecessária, o que gasta mais dados de internet e é mais lento devido a latência (Hartig e Pérez, 2018).

Por exemplo, no **Veredito**, é permitido votar de maneira positiva ou negativa em um argumento. Quando um voto é registrado, poderíamos atualizar imediatamente a interface de usuário com base nesse voto antes mesmo de recebermos confirmação do servidor. No entanto, é mais acurado aguardar a confirmação do banco de dados para obter o valor mais atualizado, uma vez que outro usuário pode ter votado simultaneamente, o que alteraria o resultado final.

Com GraphQL, definimos uma mutação (operação usada para modificar dados) responsável por atualizar um argumento do lado do servidor, e então no frontend, solicitamos como retorno da mutação apenas a quantidade de votos, supondo que as outras informações como autor e corpo não mudaram. Isso economiza bastante tráfego de rede e permite uma interface mais responsiva.

Modelo de dados em GraphQL

O conjunto de dados que é disponibilizado por meio de uma interface GraphQL pode ser acessado em termos de um modelo de dados (*schema*) baseado no conjunto de dados. Esse modelo de dados é representado implicitamente como um multigrafo dirigido com nós tipados e com propriedades nos nós. Esses nós correspondem a objetos no estilo JSON que podem aparecer nos resultados de busca.

---

[4] TypeScript. https://www.typescriptlang.org/

[5] GraphQL. https://graphql.org/

Um tipo caracteriza quais campos um objeto de determinado tipo pode ter e quais são os valores permitidos para cada um desses campos. Os valores possíveis podem se restringir a um tipo específico de valores escalares ou objetos complexos (Hartig e Pérez, 2018).

Prisma

Para interagir com o sistema gerenciador de banco de dados, o PostgreSQL, usamos o Prisma[6], um ORM (*object-relational mapping*) para Node.js. O Prisma lê o modelo do banco de dados configurado e, por meio de introspecção, gera código em TypeScript pronto para ser usado por uma aplicação.

Ao importar a biblioteca em nossa aplicação, não há necessidade de escrever código em SQL para interagir com o banco de dados. Basta acessar o objeto correspondente a tabela em que se deseja fazer uma consulta ou alteração e chamar o método correspondente a operação que se deseja fazer.

1.3 Tecnologias do *frontend*

Nesta seção será feito uma breve apresentação conceitual sobre as tecnologias utilizadas na construção do *frontend* do **Veredito**.

1.3.1 *Single-page application*

A tecnologia usada na web para estruturar o conteúdo das páginas, o HTML, foi pensada inicialmente para compartilhar informações no formato de documentos. Com a popularização e sucesso da web, o HTML passou a ser usado para construir aplicações complexas que trouxeram diversas evoluções a esses formatos.

Para melhorar a interatividade dessas aplicações, surgiu o conceito de *single-page application* (SPA), em que os sites não precisam recarregar a página quando o usuário navega ou interage com o servidor. Com o lançamento do Angular.js e Backbone.js em 2010, aplicações web puderam se tornar mais dinâmicas e servir conteúdo mais rico aos usuários finais (Huotala *et al.*, 2022).

No entanto, essa abordagem introduziu problemas de desempenho, de acessibilidade e de otimização de mecanismos de busca, conjunto de técnicas conhecido como SEO (*search engine optimization*, na sigla em inglês). Em alguns casos, rastreadores da *web* eram incapazes de indexar SPAs corretamente.

Para resolver esse problema, surgiu o conceito de aplicações web isomórficas, que combina as melhores partes dos sites estáticos e dos SPAs. Uma aplicação web isomórfica primeiro renderiza a página inicial do lado do servidor e então envia para o navegador. Então, o navegador conecta manipuladores de evento e DOM (*document object model*), processo conhecido como hidratação. Após hidratar a aplicação no cliente, a aplicação se comporta como um SPA.

[6] Prisma. https://www.prisma.io/

Isso ocorre de maneira transparente para o usuário, que sequer é capaz de notar que a aplicação mudou de uma página estática para uma aplicação interativa em JavaScript (Huotala *et al.*, 2022).

1.3.2 React

O React[7] é uma biblioteca de JavaScript que permite o desenvolvimento de aplicações web interativas. Ao invés de atualizar manualmente a interface gráfica por meio da manipulação do DOM (*document object model*), o React permite que a interface seja definida de maneira declarativa, o que torna o código mais previsível e mais fácil de ser depurado. Além disso, ele também facilita o encapsulamento de código por meio de componentes, cuja composição permite a construção de interfaces complexas (Meta Platforms, 2023).

A biblioteca React é baseada em três princípios (Bugl, 2019):

1. **Declarativo**: contrário a abordagem imperativa, que descreve **como** fazer algo, em React nós descrevemos **o que** queremos fazer, como deve ser o resultado final. A biblioteca é responsável por descobrir o que deve ser atualizado na interface para atingir o resultado esperado.
2. **Baseado em componentes**: o React encapsula componentes que gerenciam seu próprio estado e suas visualizações. Então, nos é permitido compô-las a fim de criar interfaces de usuário complexas.
3. **Baixo acoplamento a outras tecnologias**: diferentemente de um arcabouço, o React não faz suposições sobre qual deve ser a sua pilha de tecnologia. Ele tem como objetivo garantir que seja possível desenvolver aplicativos rescrevendo o mínimo possível de código existente.

React Server Components

O React Server Components (RSC) é uma nova arquitetura de aplicações isomórficas projetada pela equipe do React. Ao invés de rodar do lado do cliente como um componente React normal, esses componentes são renderizados em tempo de compilação ou no servidor no momento da requisição. Isso é vantajoso porque exclui código JavaScript no pacote entregue ao cliente, de modo a diminuir o tempo de espera até que a página fique pronta.

Nem todos os componentes de uma página são dinâmicos e alguns componentes dinâmicos possuem um estado inicial previsível. O RSC se alavanca disso para fornecer uma experiência mais fluída com carregamento de interfaces complexas instantaneamente (Markbage, 2020).

Diferentemente das outras abordagens para a construção de aplicações isomórficas, o RSC permite usar a mesma base de código para fazer pré-renderização e geração estática sem configuração adicional.

[7] React. https://react.dev/

Next.js

O Next.js[8] é um arcabouço baseado em React que tem como objetivo abstrair e automaticamente configurar ferramentas necessárias para desenvolver em React de maneira produtiva como empacotamento e compilação. As principais funcionalidades do Next.js são:

1. **Roteamento**: ao invés de configurar as rotas para sua aplicação, ela é descrita implicitamente pela estrutura de arquivos do projeto.
2. **Renderização**: é possível renderizar uma interface tanto do lado do cliente quanto do lado do servidor.
3. **Aquisição de dados**: simplificada com uma API para memoização de pedidos, cacheamento de dados e revalidação.

Uma pesquisa de Patel (2023) verificou que a implementação do arcabouço Next.js resultou em soluções efetivas para melhoria do SEO do *site*, experiência do usuário melhorada devido a maior rapidez no carregamento e nas interações e aumento de tráfico orgânico e receita.

1.4 Classificação de texto

Com a popularização das tecnologias da internet, a informação em textos digitais teve um grande crescimento. Minerar efetivamente informações úteis é um grande desafio da ciência e tecnologia da informação. Uma abordagem fundamental para o processamento massivo de textos é a classificação de texto, que pode auxiliar usuários a localizar de maneira conveniente os recursos desejados e aumentar a eficiência do uso dos dados (Yao *et al.*, 2020).

No contexto deste projeto, queremos criar um classificador de texto que seja capaz de prever quais rótulos seriam mais apropriados para um tópico de discussão.

1.4.1 Definição

Seja a descrição $d \in \mathbb{X}$ de um documento, em que $\mathbb{X}$ é o espaço de documentos, e um conjunto de classes $\mathbb{C} = \{c_1, c_2, \ldots, c_j\}$. Classes também são chamadas de categorias ou rótulos. Seja $\mathbb{D}$ um conjunto de treinamento de documentos rotulados (d, c), em que $(d, c) \in \mathbb{X} \times \mathbb{C}$.

O problema da classificação de texto consiste em usar um método de aprendizado ou um algoritmo de aprendizado para obter um classificador ou uma função de classificação γ que mapeia documentos em classes:

$$\gamma : \mathbb{X} \to \mathbb{C}$$

[8] Next.js. https://nextjs.org/

Esse tipo de aprendizado é chamado de supervisionado porque há um supervisor humano responsável por definir as classes e rotular os documentos de treinamento. O método de aprendizado supervisionado é denotado por Γ e escrevemos $\Gamma(\mathbb{D}) = \gamma$, o que indica que o método de aprendizado Γ recebe o conjunto de treinamento D como entrada e retorna a função de classificação aprendida γ (Manning, Raghavan *et al.*, 2008).

1.4.2 Métodos

Ferramentas comumente usados para classificação de texto incluem o uso de modelos baseados em vetores de palavras, como o Word2Vec e o GloVe, combinados com algoritmos de classificação como o *Vector Space Model*, *Naïve Bayes*, *Support Vector Machine* e redes neurais. Esses métodos exploram a representação semântica das palavras e a estrutura do texto para aprender padrões.

O fastText[9] é uma ferramenta para obter vetores de palavras e classificar texto criada pelo Facebook Research em 2016. Essa ferramenta utiliza uma arquitetura de rede neural que incorpora representações de subpalavras, o que permite que o modelo capture informações não apenas semânticas, mas também morfológicas. Além disso, é possível usar o n-gramas das palavras, o que permite usar a ordem das palavras numa frase até certo grau, de modo a permitir uma representação mais precisa e contextual da frase.

Escolhido para classificar os tópicos do **Veredito**, o fastText utiliza regressão polinomial logística (softmax) para prever o rótulo do texto de entrada. O softmax é usado de forma hierárquica, o que reduz enormemente o tempo de treinamento de um modelo (Yao *et al.*, 2020).

1.4.3 Raspagem de dados

Para treinar um modelo de classificação de texto, é necessário uma grande quantidade de dados de treino e de validação. Existem diversas técnicas disponíveis para a extração de dados de sites para diversas finalidades. Tipicamente, os dados encontrados na web não são estruturados, então é necessário uma transformação para que eles sejam armazenados e consultados.

Dentre as diversas técnicas para raspagem de dados, podemos citar as principais (Sirisuriya, 2015):

1. Copia e cola manual: esse é o método mais simples e ocasionalmente, é suficientemente funcional para os objetivos. Mas essa técnica é sujeita a erros, entediante e cansativa quando as pessoas precisam extrair conjuntos de dados muito grandes.
2. Automação via HTTP: essa técnica pode ser usada para extrair dados de páginas estáticas e dinâmicas. Os dados podem ser obtidos ao enviar requisições para um servidor de maneira automatizada.
3. Navegação sintética: por meio de software de automações que controlam navegadores como o Mozilla Firefox ou Google Chrome, é possível executar uma sequência de atividades em *sites* e extrair os conteúdos. Apesar dessa forma ser mais complexa,

[9] fastText. https://fasttext.cc/

ela tem menos chance de falhar ou de produzir resultados diferentes por ser a que mais se aproxima de um ser humano real.

A técnica ideal varia de acordo com os objetivos e características das fontes de dados. Ferramentas como o jq podem auxiliar na extração de dados em JSON (formato comumente usado na *web*) e aplicações de planilhas podem ser úteis para visualizar e análises rápidas dos dados extraídos. Naturalmente, análises mais complexas podem exigir o uso de alguma linguagem de programação, como Python, que é popular na área de ciência de dados.

1.4.4 ChatGPT

O ChatGPT[10] é um sistema de processamento de linguagem natural (*natural language processing* ou NLP) desenvolvido pela Open AI. Ele é projetado para gerar conversas humanizadas ao entender o contexto de um diálogo e gerar respostas apropriadas. O ChatGPT é baseado em um modelo de aprendizagem profunda chamado GPT-3, que é treinado em um conjunto de dados muito grande de conversas.

O ChatGPT utiliza o modelo *ChatGPT Improved Accuracy* (CGA), que é capaz de gerar conversas realistas e interessantes, cujas habilidades generativas são melhoradas pela sua capacidade de aprender com os próprios erros. Isso permite que o CGA se adapte a novos contextos e produza resultados mais precisos. Comparado a outros modelos populares de NLP, o CGA tem um desempenho impressionante e é muito superior em termos de precisão, coerência e legibilidade (Deng e Lin, 2022).

Neste trabalho, o ChatGPT foi usado para re-classificar os tópicos de discussão do Kialo. Essa nova classificação foi útil para obter dados de treinamento mais padronizados, de modo a melhorar o desempenho do classificador.

[10] ChatGPT. https://chat.openai.com/

Capítulo 2

Implementação

Neste capítulo vamos apresentar as decisões de projeto e detalhes sobre a implementação do **Veredito**.

2.1 Escopo e funcionalidades

O **Veredito** é uma plataforma que pretende oferecer um ambiente mais propício para o debate público em uma interface intuitiva para a criação de tópicos e argumentos e com o apoio de ferramentas de categorização e moderação por meio do histórico de mudanças.

Nessa ferramenta, as discussões são organizadas em tópicos com argumentos divididos entre prós e contras. É possível rebater cada argumento com contra-argumentos em apenas um nível de discussão, limitação que pretende evitar os problemas da estrutura em árvore discutidos na introdução, como fragmentação e perda de contexto.

Não é possível acrescentar subargumentos a favor: caso o usuário acredite que é possível complementar um argumento com dados ou outro raciocínio, é possível sugerir uma edição no próprio texto. Nesse formato é possível ter argumentos mais autocontidos, o que pode facilitar a leitura e compreensão do assunto.

Caso um ponto da discussão seja mais polêmico, é possível criar um novo tópico e referenciá-lo no argumento, de modo a usá-lo como hipótese. Assim, a comunidade pode ter discussões paralelas de assuntos tangenciais em contextos separados, sem que a plataforma perca a relação semântica entre esses assuntos.

Pretende-se também aplicar tecnologias de aprendizado de máquina e processamento de linguagem natural para facilitar as atividades de moderação. Além disso, foi usado técnicas de raspagem de dados e redes neurais para sugestão automática de rótulos para tópicos.

2.2 *Frontend*

A interface gráfica do Veredito é *web* e responsiva, de modo a funcionar tanto em computadores de mesa quanto em dispositivos móveis. Ela usa React, uma biblioteca que facilita o desenvolvimento de aplicações web interativas, e Tailwind, um arcabouço de CSS que facilita o uso de um sistema de design coerente em toda a interface.

2.2.1 Mudança de arcabouço React

Inicialmente, o projeto estava implementado utilizando o arcabouço Gatsby. No entanto, migramos para o Next.js porque o Gatsby é especializado em geração de páginas estáticas (SSG). É possível usá-lo para experiências mais interativas, mas ele oferece poucos recursos para mitigar os problemas do modelo SPA mencionados na seção 1.3.1.

Como resultado, ao fazer o provisionamento da aplicação e ao acessar um tópico no ambiente de produção, a página retornava um erro HTTP não encontrado (código 404), e só após isso exibia o conteúdo. Após algumas tentativas de resolver o problema, ficou claro que essa não era a ferramenta certa para abordar o caso em que páginas são criadas de maneira dinâmica por meio de interação do usuário.

Como os dois arcabouços são baseados em React, foi possível aproveitar boa parte do código na migração. No entanto, o mecanismo de obtenção de dados da API teve que ser completamente refeito. Isso porque o *React Server Components* (RSC) que renderiza páginas no servidor antes de entregar ao cliente não tem estado e, portanto, não pode usar os *hooks* do React, de modo a tornar a biblioteca cliente de GraphQL chamada RTK Query incompatível.

Como se trata de uma aplicação isomórfica, isto é, a renderização de certos componentes é realizada de maneira intercambiável no *frontend* e no *backend*, optamos por remover o RTK Query também dos componentes do cliente para que o código entre RSC e componentes do cliente pudesse ser reaproveitado entre si. A biblioteca usada para isso foi a `graphql-request`, que é a maneira mais simples e mínima de usar um cliente de GraphQL.

Isso teve algumas implicações no desenvolvimento. Por exemplo, o RTK Query fazia controle de estado automaticamente por meio de React *hooks*, então era muito conveniente sinalizar que o componente estava sendo carregado com uma animação de carregamento. Já com o `graphql-request` todo esse controle precisa ser feito manualmente.

2.2.2 Lista de discussões

A lista de discussões (figura 2.1) corresponde a página inicial do projeto. Nela, é possível ver o título dos tópicos e a quantidade de votos e argumentos. Esses números são relevantes para que o usuário seja capaz de avaliar quais são as discussões populares e com mais atividade sem ter que navegar.

Além disso, é possível ver rótulos em cada item, o que pode ajudar a dar contexto sobre em qual tema aquela discussão se encaixa.

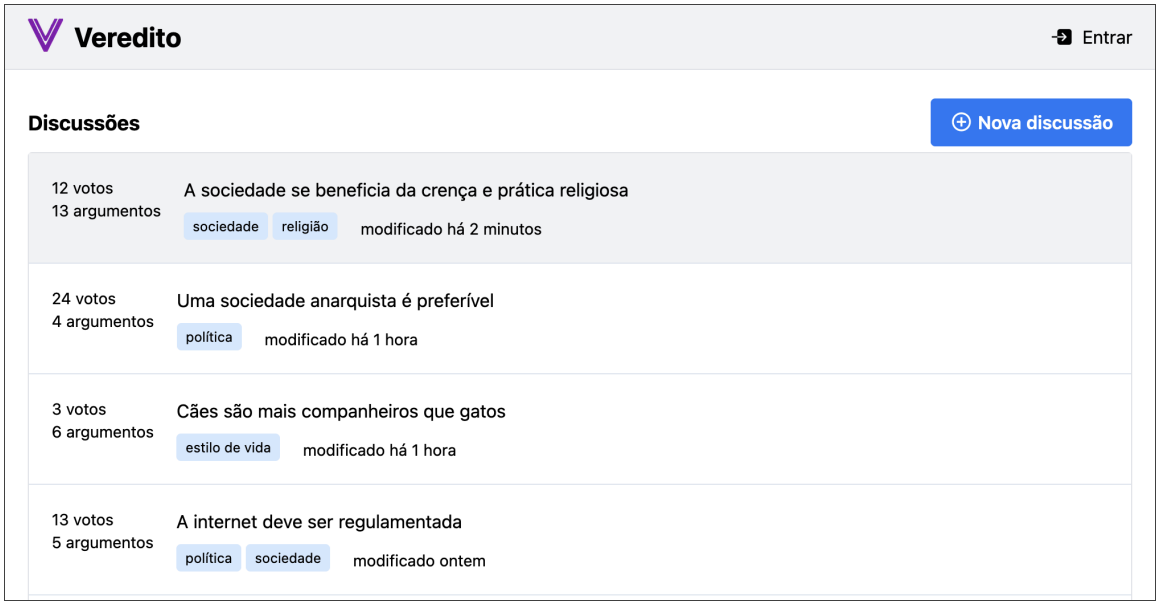

Figura 2.1: *Lista de discussões*

2.2.3 Página de discussão

A página de discussão (figura 2.2) é composta por alguns detalhes do tópico em debate, como título e datas de criação e modificação, e segue com os argumentos favoráveis (posicionados à esquerda) e contrários (posicionados à direita).

Inspirado pelo Stack Overflow[1], a caixa de texto para criação de um novo argumento sempre fica disponível, de modo a incentivar a participação. É permitido criar argumentos sem estar autenticado, mas o endereço de IP fica associado àquela alteração e pode ser verificado por qualquer um no histórico. Essa abordagem é inspirada pela Wikipédia[2] e também tem como objetivo incentivar a colaboração pública, pois criar uma conta pode gerar um atrito que afaste a colaboração.

Os votos, no entanto, são restritos a usuários registrados. Isso é necessário porque caso contrário, uma pessoa poderia votar quantas vezes quisesse. Seria possível restringir um voto por endereço de IP, mas isso prejudicaria pessoas que usam redes compartilhadas (por exemplo, todas as pessoas que usam o Wi-Fi de uma universidade só teriam direito a um voto).

Existem alguns botões no rodapé de cada argumento:

1. Discussão: permite discutir sobre o próprio argumento. Trata-se de um espaço para meta-discussão que pode ser usado para melhorar um argumento ou para mostrar uma fraqueza dele. Se for de interesse da comunidade debater um argumento como um tópico, o ideal é criar um tópico para discuti-lo. No futuro poderá ser possível referenciar tópicos dentro dos argumentos para usá-los como premissas.

[1] Stack Overflow - https://stackoverflow.com/

[2] Wikipedia - https://meta.wikimedia.org/wiki/Unregistered_user

2. Histórico: exibe a data, descrição e usuário ou IP que fez alguma alteração no argumento. É possível expandir e ver como era o argumento antes daquela edição.
3. Editar: permite que o argumento seja editado após fornecer uma descrição da edição.
4. Compartilhar: permite copiar um *link* que referencia aquele argumento dentro da discussão.

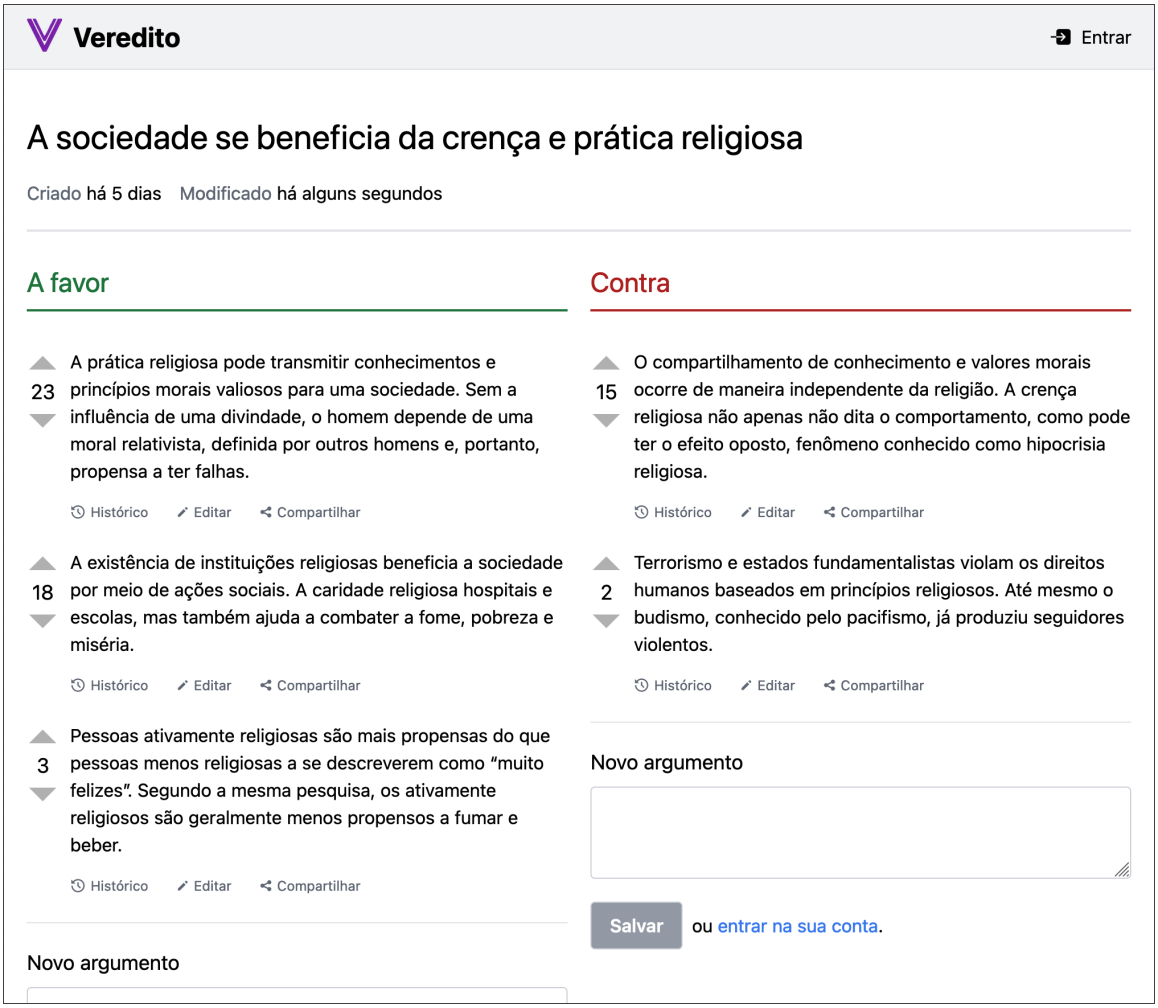

Figura 2.2: *Página de discussão*

2.3 *Backend*

2.3.1 Modelo de dados

Os dados na plataforma são representados de acordo com o paradigma relacional, cujo esquema é dado pela figura 2.4. Foi utilizado o sistema gerenciador de banco de dados PostgreSQL.

A interação na plataforma é dada por meio de tópicos, que podem conter argumentos prós ou contras. Como há muitas relações e atributos em comum entre tópicos e argumentos,

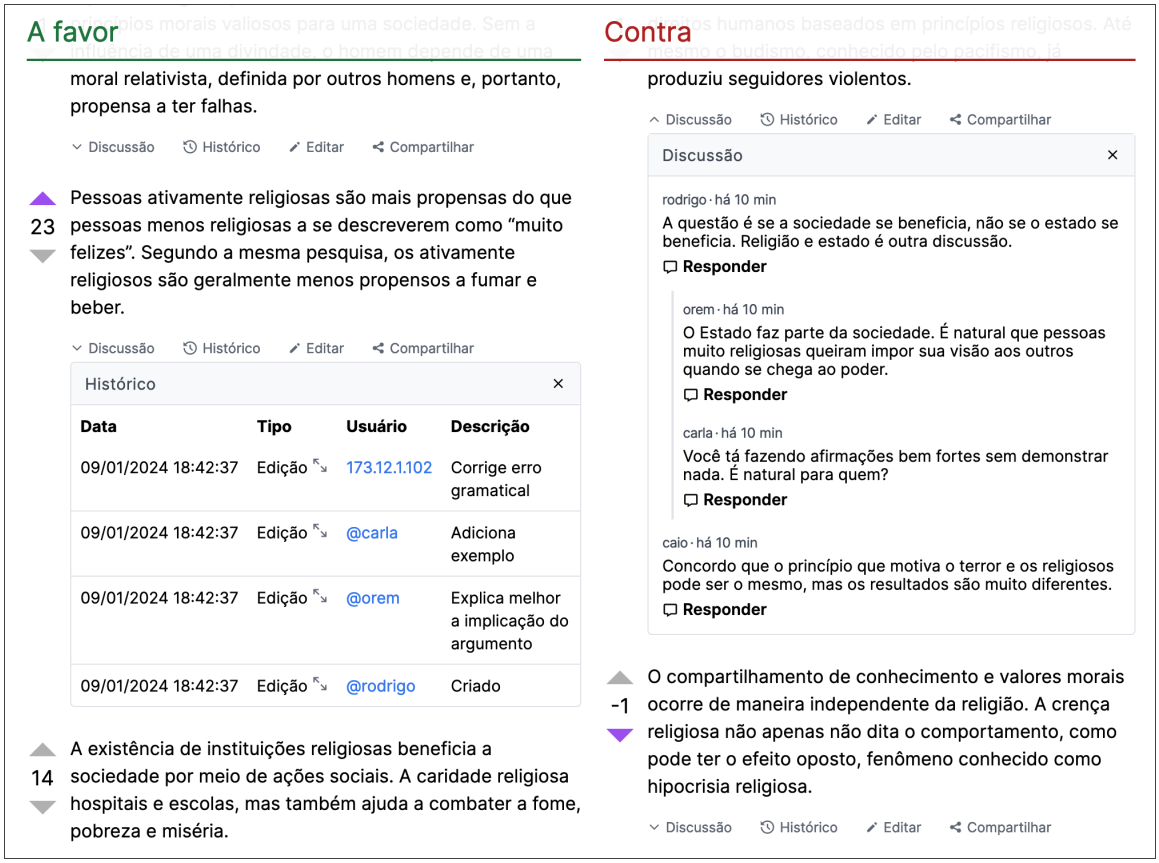

Figura 2.3: *Página de discussão com histórico e discussão*

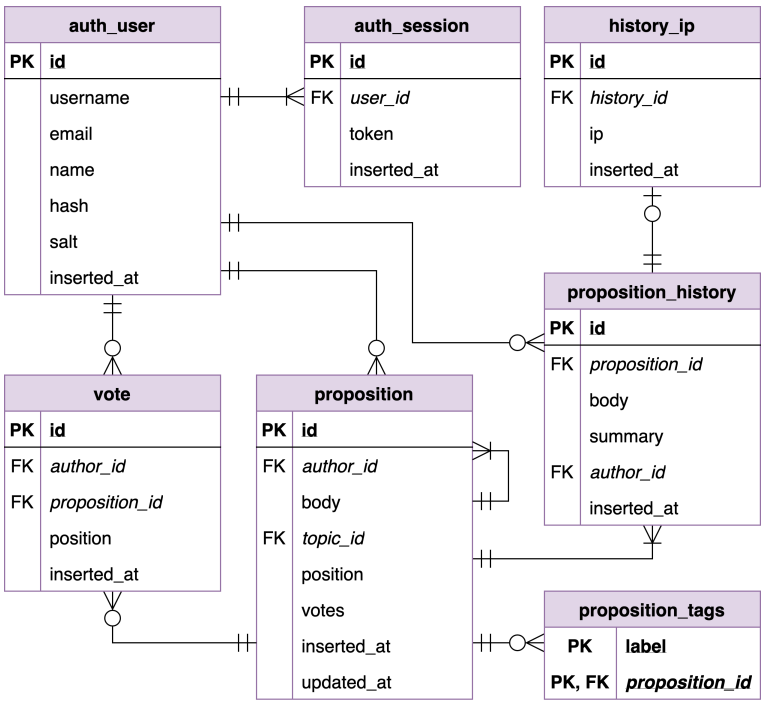

Figura 2.4: *Diagrama do modelo de dados do Veredito*

os dois são representados pela mesma entidade *proposition*. É possível diferenciá-los pelo atributo *topic_id*, afinal, argumentos referenciam a *proposition* correspondente ao tópico que eles pertencem, enquanto tópicos têm esse campo em branco.

Todas as edições são armazenadas em *proposition_history*, de modo a permitir o desfazimento de uma edição que piorou ou vandalizou o conteúdo. Quando o usuário está registrado, seu nome de usuário é associado à edição, caso contrário, o endereço de IP também é salvo.

2.3.2 Autenticação

Como o protocolo HTTP não tem estado, a autenticação em sistemas *web* deve ser implementada a nível de aplicação. O **Veredito** implementa a abordagem tradicional, baseada em sessão, que consiste nos seguintes passos:

1. Quando o usuário se registra na plataforma, o *hash* criptográfico da sua senha é armazenado no banco de dados.

2. Ao se autenticar com sucesso na plataforma, é gerado uma sequência aleatória que identifica sua sessão e tem um prazo de validade, o *token* de sessão. Esse *token* é armazenado no *cookie* de seu navegador.

3. Em requisições subsequentes, o *cookie* com a sessão é enviado para o servidor, que verifica se a sessão é válida. Caso não corresponda a nenhuma sessão criada anteriormente ou seja uma sessão expirada, a requisição não é autorizada.

O *hash* criptográfico criado no passo 1 é a concatenação da senha do usuário e uma sequência aleatória de caracteres, chamada de *salt*, que também é armazenado no banco de dados. Ao autenticar o usuário, é necessário gerar o *hash* da concatenação da senha e do *salt* e comparar com os dados armazenados no banco de dados.

O *salt* é importante para inviabilizar ataques de dicionário ou de tabelas arco-íris, que usam valores pré-computados de senhas comuns para descobrir senhas dos usuários. Esses tipos de ataques ocorrem quando um invasor obtém acesso ao banco de dados, de modo que o uso de *salt* mantém a senha do usuário protegida mesmo nesse caso. Isso é importante considerando que muitos usuários usam a mesma senha em vários serviços.

Outra preocupação que se deve ter ao implementar um serviço de autenticação é o ataque de temporização. Esse ataque consiste em descobrir qual posição de caractere de um *token* está incorreta baseado no tempo de resposta do servidor. Se os primeiros caracteres não forem correspondentes ao *token* legítimo, um algoritmo vulnerável retornaria o erro de autenticação alguns milisegundos mais rápido do que se os caracteres errados estiverem no final do *token*. Isso pode dar uma pista para um invasor escolher permutações de maneira mais eficiente do que com força bruta. Para inviabilizar esse tipo de ataque, é necessário implementar um algoritmo que verifica se os *tokens* são iguais em tempo constante.

Para evitar ataques de injeção de conteúdo (XSS), é recomendado usar *cookies HttpOnly* (Calzavara *et al.*, 2017). Esse mecanismo impede que qualquer código executado no navegador roube o código de autenticação. Mas como o *frontend* e o *backend* são projetos separados que rodam em domínios separados, é necessário fazer configurações adicionais

para permitir que o código de sessão trafegue normalmente de maneira segura. Essas configurações envolvem definir cabeçalhos HTTP de CORS (*cross origin resource sharing*) e atributos específicos tanto nas configurações do *cookie*, quanto nas configurações da API *fetch*, responsável pelas requisições.

Para conveniência, o mecanismo que verifica se um usuário está registrado é fornecido para os outros componentes do projeto como um "guarda de autenticação". Esse guarda usa um padrão de projeto chamado decorador, o que permite que qualquer rota da API exija autenticação facilmente ao aplicar o decorador na função.

Capítulo 3

Classificação de tópicos

Uma das funcionalidades do **Veredito** é a classificação automática de tópico de discussão em rótulos (*tags*). Isso facilita a padronização e a organização da plataforma, o que pode permitir que os usuários encontrem conteúdo relevante com mais facilidade.

Treinamos um modelo de aprendizado de máquina para fazer a classificação automática desses tópicos de discussão. Os dados usados no treinamento foram extraídos do Kialo, que já possui debates classificados com rótulos.

Após análise dos dados, notou-se que poderia ser necessário uma classificação mais padronizada para treinar o modelo. Utilizamos, portanto, o ChatGPT para re-classificar os tópicos obtidos do Kialo e diminuir o número de rótulos utilizados no treinamento, a fim de melhorar o desempenho do modelo.

3.1 Raspagem de dados

Ao navegar pela lista de debates no Kialo, é possível encontrar, com a ajuda de um inspetor de tráfego *web*, a requisição para uma API que retorna esses dados. Essa requisição realiza uma chamada HTTP *GET* na seguinte URL (formatada para facilitar a visualização):

```
https://www.kialo.com/api/v1/discussions?
    filter=promoted
    &sort=view_count
    &limit=24
    &skip=24
    &timestamp=1708946424594
    &dfuisadhr0n4g=0r4sxu
```

Nota-se que, além do filtro e da ordenação, existe um mecanismo de paginação por meio dos parâmetros *limit* e *skip*. Com isso, é possível automatizar a extração de uma grande quantidade de dados de maneira simples usando uma ferramenta de linha de comando, como o *curl*.

Neste trabalho, usamos como parâmetro de *limit* 200, o que significa que 200 itens de tópicos de discussão serão retornados a cada requisição. Para extrair os dados, basta variar o parâmetro *skip*, começando em zero e incrementando de 200 em 200. O formato do tópico, em JSON, está representado no Programa 3.1. Após uma sequência de 16 requisições bem sucedidas, a API retorna HTTP 204 (código que significa que a página não tem conteúdo). Isso indica que o limite de resultados retornado por essa API foi atingido.

```json
{
  "id": 16404,
  "title": "Do the ends justify the means?",
  "claimAndThesisCount": 201,
  "viewCount": 7774,
  "participantCount": 75,
  "contributionCount": 826,
  "voteCount": 467,
  "created": 1530171056077,
  "tags": [
    "Morality",
    "Philosophy",
    "Justice"
  ],
  "isPublic": true,
  "isArchived": false,
  "latestActivity": {
    "discussionIdentityId": "5baf246f4c43c805276a5337",
    "type": "admin-replied-suggested-claim",
    "date": 1679754171050
  },
  "activitiesLastDay": 0,
  "activitiesLastWeek": 0,
  "image": "e76fc468-dd96-4dee-9069-43fb63d990a0",
  "language": "en",
  "featured": null,
  "popular": false,
  "trending": false,
  "rank": null,
  "lastSeen": null,
  "effectiveRole": "suggester",
  "followed": "not-set",
  "isFresh": false,
  "accessToken": null,
  "isMultipleChoice": false
}
```

Programa 3.1: *Exemplo de tópico em JSON extraído do Kialo*

Com isso, os arquivos JSON associados a cada tópico foram armazenados em um

diretório e a ferramenta de linha de comando `jq` foi utilizada para juntar todos os arquivos e transformá-los em uma tabela no formato CSV, o que facilita a análise de dados. Os rótulos, que são armazenados em uma lista em JSON, foram concatenados e separados por vírgula, de modo a pertencerem a uma mesma coluna da tabela em CSV.

3.1.1 Análise dos dados

A extração de dados resultou em um total de 2.677 tópicos de debate. Existem tópicos em 25 idiomas, mas os tópicos em inglês são majoritários, correspondendo a 90,32% do total (2.417 tópicos). As outras linguagens com maior incidência foram alemão (2,19%), espanhol (2,07%) e francês (1,78%).

Existem dados de popularidade, de votos e de contribuidores, que poderiam ser úteis no futuro para dar um peso maior a certos itens no treinamento do modelo. Outra coluna de interesse é a data da última modificação, que pode ser útil para estimar o quão recente é cada discussão. A coluna que será objeto de estudo neste trabalho, contudo, é a de rótulos dos tópicos.

Rótulos

Considerando apenas os 2.417 tópicos em inglês, há 2.015 rótulos únicos. A figura 3.1 mostra a distribuição dos 10 rótulos com maior incidência. O 10 rótulos mais usados (0,4% dos rótulos) são usados em 57% de todos os tópicos do Kialo.

Essa concentração de poucos rótulos em muitos tópicos pode ser observada no gráfico da figura 3.2. Como mostra a cauda longa do gráfico, a maioria dos rótulos é bem dispersa pois é usada em poucos tópicos. De fato, 69,6% dos rótulos são usados em menos de 10 tópicos.

No gráfico da figura 3.3, é possível observar a frequência da quantidade de rótulos por tópico. A mediana e a moda são de 4 rótulos por tópico.

A alta dispersão e a alta quantidade de rótulos por tópicos pode ser problemática para treinar um modelo de classificação de texto, pois existem muitas respostas possíveis para o rótulo de um tópico e muito mais categorias do que o necessário.

3.2 Classificação usando o ChatGPT

Como os rótulos serão utilizados para treinar um modelo de classificação, será necessário obter outra classificação de treinamento com menos rótulos e mais padronizada. Portanto, o ChatGPT 3.5 será usado para re-classificar os tópicos extraídos do Kialo.

A partir da lista dos rótulos mais usados, foi feita uma análise manual para estabelecer uma lista com 37 categorias que seriam utilizadas nessa nova classificação. Por meio da API do ChatGPT, foi construído um *script* simples capaz de enviar cada tópico com o seguinte *prompt* (traduzido do inglês):

> Você é um classificador de texto, responda apenas com as seguintes *tags* permitidas: Academia, Arqueologia, Arte, Negócios, Computação, Cultura,

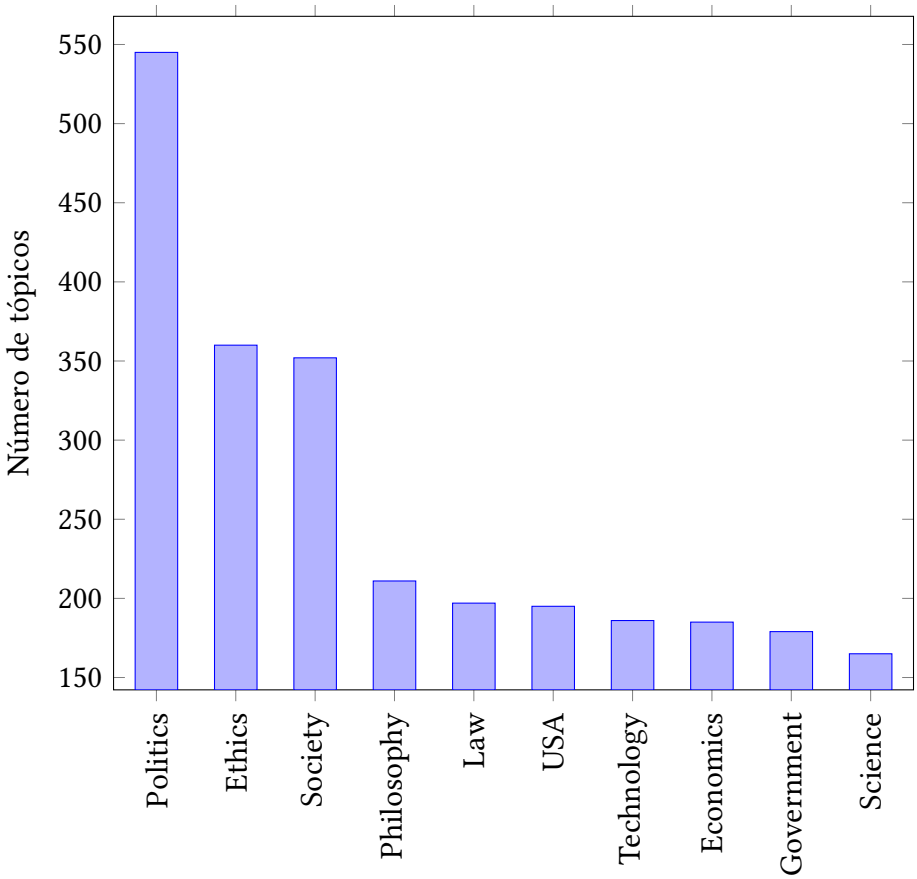

Figura 3.1: *Distribuição dos 10 rótulos mais frequentes*

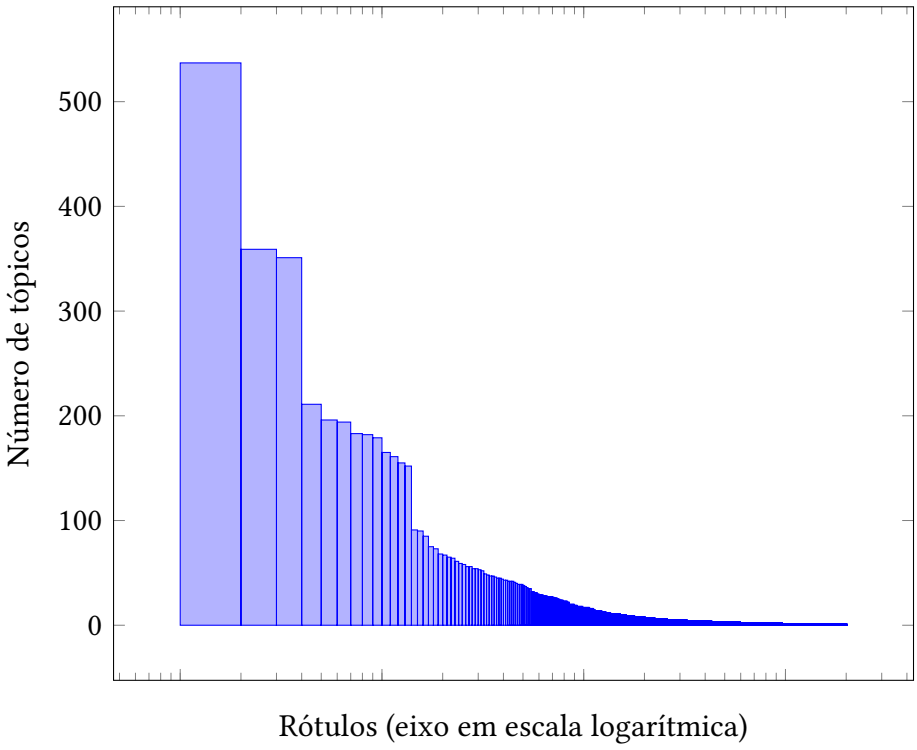

Figura 3.2: *Distribuição de todos os rótulos*

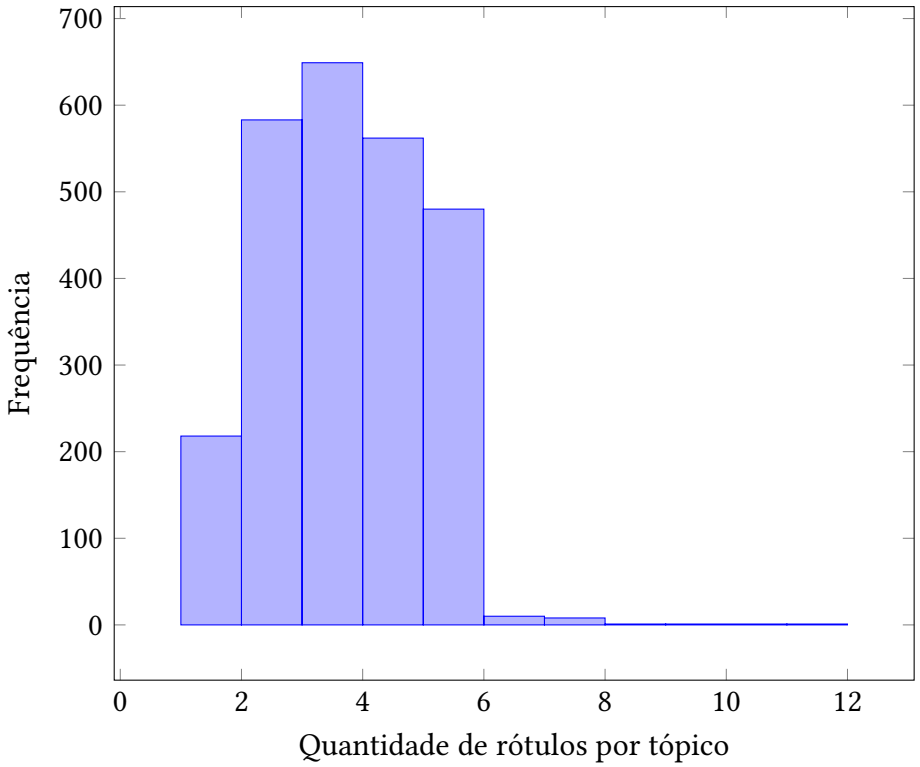

Figura 3.3: *Distribuição do número de rótulos por tópico de discussão*

> Economia, Educação, Meio Ambiente, Ética, Finanças, Gênero, Questões Globais, Governança, Saúde, História, Direitos Humanos, Inovação , Jurídico, Literatura, Gestão, Mídia, Medicina, Filosofia, Política, Psicologia, Regulamentação, Religião, Pesquisa, Ciência, Sociedade, Sociologia, Espiritualidade, Esportes, Sustentabilidade, Tecnologia. Use no máximo 4 *tags* por resposta.

No entanto, como os tópicos têm o formato de pergunta, frequentemente o ChatGPT não respondia com o rótulo (*tag*) correspondente, e sim respondendo a própria pergunta. Isso foi mitigado ao preceder cada tópico com o seguinte "Classifique o seguinte tópico de discussão".

3.3 Classificação usando modelo treinado com fastText

Foi utilizado o fastText para produzir um modelo de classificação de texto. Para isso, os tópicos de discussão e seus rótulos foram divididos em um conjunto de 80% para treinamento e de 20% para validação. Foram testados os resultados para a classificação usando dados de treinamento do próprio Kialo e também do Kialo classificado pelo ChatGPT.

Os resultados dos modelos de classificação (tabela 3.1) nos mostram que o modelo que usou a classificação do ChatGPT como dados de treinamento teve um desempenho muito superior ao modelo com os dados do Kialo. Os resultados comparados são o de precisão em 1, que mede a proporção das amostras corretamente classificadas pelo modelo,

	Fonte dos dados usados no treinamento	
	Kialo	ChatGPT
Precisão em 1	14,1%	56%
Recall em 1	3,9%	26,8%
F1-Score	6,11%	36,25%

Tabela 3.1: *Desempenho dos modelos de classificação*

considerando apenas a classe principal atribuída a cada amostra.

O modelo treinado com dados de categorização do Kialo acertou a classe principal em 14,1% dos tópicos do conjunto de teste. Já o modelo treinado com dados do Kialo re-classificados pelo ChatGPT acertou a classe de 56% das amostras do conjunto de teste, o que é uma taxa de acerto 3,97 vezes superior.

Com investigação no conjunto de dados da classificação do Kialo, descobriu-se que muitos rótulos no conjunto de validação sequer apareciam no conjunto de treinamento. Isso se deve a alta dispersão dos rótulos e ao fato de que muitos rótulos são usados poucas vezes na categorização do Kialo. Isso é problemático porque no caso de rótulos que são usados apenas uma vez (7,4% dos rótulos), é impossível que o classificador acerte a resposta se o tópico estiver no conjunto de validação.

Esse problema poderia ter sido reduzido se o desbalanceamento dos rótulos fosse levado em conta na separação dos dados para treinamento e avaliação. É desejável que o conjunto de treinamento tenha classes o mais balanceadas possível, o que é um desafio já que o conjunto de dados disponível não era suficientemente grande.

3.3.1 Parâmetros

É possível utilizar alguns parâmetros ao treinar um modelo no fastText. O comando utilizado para treinar o modelo foi:

```
fasttext supervised -input kialo.train -output model_kialo -lr 0.5 -epoch 25
    -wordNgrams 2 -bucket 200000 -dim 50 -loss one-vs-all
```

Os valores escolhidos para cada parâmetro foram inspirados no tutorial da ferramenta[1] e ajustados com breves testes empíricos. Os parâmetros são:

1. `lr` (taxa de aprendizado): Corresponde ao quanto o modelo muda após processar cada exemplo.
2. `epoch` (época): Refere-se ao número de vezes que cada exemplo é visto pelo modelo.
3. `wordNgrams` (n-gramas de palavras): Um n-grama é uma subsequência de n palavras do texto a ser processado. Preservar a sequência pode ajudar a capturar o contexto dentro de um texto.

[1] https://fasttext.cc/docs/en/supervised-tutorial.html

4. `bucket` (entradas): O recurso de n-gramas é implementado com um *hash* em um número fixo de entradas, a fim de limitar o uso de memória do modelo. Essa opção controla o número de entradas usadas.
5. `dim` (dimensão): controla o tamanho dos vetores, quanto maior mais informação eles guardam mas requer mais dados para serem aprendidos.
6. `loss` (função de perda): trata-se de uma função usada no algoritmo *softmax*. Com o valor `one-vs-all`, é possível atribuir múltiplos rótulos por meio do uso de classificadores binários para cada rótulo.

Capítulo 4

Conclusão

Este trabalho construiu um sistema *web* e pôde explorar tecnologias de desenvolvimento de sistemas, padrões arquiteturais, mineração de dados e um modelo de classificação de texto. Além disso, foi possível pesquisar sobre o domínio do projeto, que são os problemas sociais do debate *online*. Isso possibilitou a proposta de soluções de interesse do estudo da computação que também abordassem problemas sociais.

A implementação inicial do *frontend* do sistema foi feita em Gatsby, escolhido para facilitar o desenvolvimento e melhorar o desempenho. No entanto, como o Gatsby é especializado em geração de páginas estáticas, há pouco suporte para casos em que as páginas são geradas automaticamente – como é o caso das páginas de discussão do **Veredito**. Isso fez com que o *frontend* fosse reimplementado em Next.js, arcabouço projetado para aplicações com conteúdo dinâmico.

A reimplementação foi interessante para perceber as diferenças entre os dois arcabouços e verificar as dificuldade de uma migração desse tipo, considerando que os dois arcabouços usam a biblioteca React. Muito código pôde ser reaproveitado, mas a grande fonte de incompatibilidade que exigiu a rescrita completa é o mecanismo de obtenção de dados (*data fetching*).

O conceito de aplicações isomórficas, em que uma aplicação pode ser renderizada tanto no cliente quanto no servidor, dependendo do que for mais rápido para o usuário, está presente na *web* desde 2010. No entanto, em React não há uma forma padrão de disponibilizar os dados obtidos por uma API para a interface de usuário que funcione tanto no cliente quanto no servidor.

A maior dificuldade em fazer isso é que no cliente há necessidade de fazer controle de estado (por exemplo, exibir uma animação de carregamento enquanto os dados são obtidos, ou esconder certo itens se o usuário não está registrado). Já no servidor, não há estado: a página é fornecida de maneira completa, com os dados já presentes. Isso torna o desenvolvimento de aplicações isomórficas trabalhoso e com duplicação de código. Acreditamos que há oportunidade para melhoria do fluxo de desenvolvimento nessa área por meio de bibliotecas que façam a gestão da obtenção de dados de maneira realmente isomórfica.

No *backend*, notou-se que a adoção de GraphQL foi bem vantajosa. Apesar da implementação inicial ser um pouco mais complexa que uma API REST, o GraphQL oferece muita flexibilidade ao consultar os dados no *frontend*. Como é possível agregar o conteúdo de duas ou mais entidades em uma única requisição, isso levou a uma quantidade de código menor no *frontend* e poucos ajustes na API do *backend* após a implementação inicial.

Em relação a classificação de texto, um desafio abordado foi a despradonização de rótulos usados na plataforma de debates Kialo. É difícil treinar um modelo de classificação com dados tão dispersos, porque isso significa que os dados não estão bem classificados. Para evitar isso, seria interessante limitar as escolhas do usuário em relação ao rótulo.

O ChatGPT é uma ferramenta muito útil e sofisticada para treinar um classificador de textos. Como a integração via API é paga, seria custoso usá-lo diretamente em um serviço para fornecer facilidades não-essenciais, como a sugestão de rótulos de tópicos de discussão. No entanto, usá-lo para treinar um modelo custa poucos dólares e produz ótimos resultados, mesmo com um conjunto de dados não tão grande. Isso foi verificado na precisão de 56% do experimento realizado neste trabalho com apenas 2.417 tópicos de discussão.

4.1 Trabalho futuro

Para validar se a solução proposta por esse trabalho é útil para outras pessoas, é importante fazer uma pesquisa de usabilidade. Essa pesquisa poderia avaliar se esta aplicação é fácil de usar e efetiva como uma plataforma de debates online.

Com o uso de técnicas de processamento de linguagem natural e aprendizado de máquina, seria possível detectar automaticamente argumentos duplicados durante a criação de um argumento. Isso seria útil para evitar a duplicação de entradas no banco de dados ao sugerir que o usuário referencie ou edite argumentos correspondentes já existentes.

Outra oportunidade de uso de inteligência artificial seria calcular a relevância de argumentos a partir da frequência com que eles são usados como premissa de outros argumentos. Seria possível, por exemplo, utilizar o algoritmo do PageRank adaptado para relevância de argumentos (Wachsmuth *et al.*, 2017).

Referências

[Berners-Lee *et al.* 1994] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen e Arthur Secret. "The world-wide web". *Communications of the ACM* 37.8 (1994), pp. 76–82 (citado na pg. 5).

[Bierman *et al.* 2014] Gavin Bierman, Martín Abadi e Mads Torgersen. "Understanding typescript". In: *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28*. Springer. 2014, pp. 257–281 (citado nas pgs. 6, 7).

[Bugl 2019] Daniel Bugl. *Learn React Hooks: Build and refactor modern React. js applications using Hooks*. Packt Publishing Ltd, 2019 (citado na pg. 9).

[Calzavara *et al.* 2017] Stefano Calzavara, Riccardo Focardi, Marco Squarcina e Mauro Tempesta. "Surviving the web: a journey into web session security". *ACM Computing Surveys (CSUR)* 50.1 (2017), pp. 1–34 (citado na pg. 18).

[Chaudoin *et al.* 2017] Stephen Chaudoin, Jacob Shapiro e Dustin Tingley. "Revolutionizing teaching and research with a structured debate platform". *Journal of Political Science* 58 (2017), pp. 1064–1082 (citado na pg. 2).

[Cheng *et al.* 2017] Justin Cheng, Michael Bernstein, Cristian Danescu-Niculescu-Mizil e Jure Leskovec. "Anyone can become a troll: causes of trolling behavior in online discussions". In: *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*. 2017, pp. 1217–1230 (citado nas pgs. 1, 2).

[Deng e Lin 2022] Jianyang Deng e Yijia Lin. "The benefits and challenges of chatgpt: an overview". *Frontiers in Computing and Intelligent Systems* 2.2 (2022), pp. 81–83 (citado na pg. 12).

[Diakopoulos e Naaman 2011] Nicholas Diakopoulos e Mor Naaman. "Towards quality discourse in online news comments". In: *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. 2011, pp. 133–142 (citado na pg. 1).

[Ehrenfeld e Barton 2019] Dan Ehrenfeld e Matt Barton. "Online public spheres in the era of fake news: implications for the composition classroom". *Computers and Composition* 54 (2019), p. 102525. issn: 8755-4615 (citado na pg. 1).

[Győrödi *et al.* 2020] Cornelia A Győrödi *et al.* "Performance analysis of nosql and relational databases with couchdb and mysql for application's data storage". *Applied Sciences* 10.23 (2020), p. 8524 (citado na pg. 6).

[Hartig e Pérez 2018] Olaf Hartig e Jorge Pérez. "Semantics and complexity of graphql". In: *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 1155–1164 (citado nas pgs. 7, 8).

[Huotala *et al.* 2022] Aleksi Huotala, Matti Luukkainen e Tommi Mikkonen. "Benefits and challenges of isomorphism in single-page applications: case study and review of gray literature". *Journal of Web Engineering* 21.8 (2022), pp. 2363–2403 (citado nas pgs. 8, 9).

[Jhaver *et al.* 2019] Shagun Jhaver, Iris Birman, Eric Gilbert e Amy Bruckman. "Human-machine collaboration for content regulation: the case of reddit automoderator". *ACM Transactions on Computer-Human Interaction (TOCHI)* 26.5 (2019), pp. 1–35 (citado nas pgs. 1, 2).

[Manning, Raghavan *et al.* 2008] Christopher D Manning, Prabhakar Raghavan *et al. Schü tze h. introduction to information retrieval*. 2008 (citado na pg. 11).

[Markbage 2020] Sebastian Markbage. *RFC: React Server Components*. 2020. URL: https://github.com/reactjs/rfcs/blob/main/text/0188-server-components.md (citado na pg. 9).

[Mcelfresh 2016] Joshua Mcelfresh. *Spirited: A Web Application for Structured Debate*. Harvard University, 2016 (citado na pg. 3).

[Meta Platforms 2023] Inc Meta Platforms. *React – a JavaScript library for building user interfaces*. 2023. URL: https://reactjs.org/ (citado na pg. 9).

[Mounk 2018] Yascha Mounk. "The people vs. democracy". In: *The People vs. Democracy*. Harvard University Press, 2018 (citado na pg. 1).

[Padhy *et al.* 2011] Rabi Prasad Padhy, Manas Ranjan Patra e Suresh Chandra Satapathy. "Rdbms to nosql: reviewing some next-generation non-relational database's". *International Journal of Advanced Engineering Science and Technologies* 11.1 (2011), pp. 15–30 (citado na pg. 6).

[Patel 2023] Vishal Patel. "Analyzing the impact of next. js on site performance and seo". *International Journal of Computer Applications Technology and Research* 12 (2023), pp. 24–27 (citado na pg. 10).

[Pavlenko *et al.* 2020] Andrey Pavlenko, Nursultan Askarbekuly, Swati Megha e Manuel Mazzara. "Micro-frontends: application of microservices to web frontends." *J. Internet Serv. Inf. Secur.* 10.2 (2020), pp. 49–66 (citado na pg. 5).

[Sirisuriya 2015] D S Sirisuriya. “A comparative study on web scraping”. In: *International Research Conference*. Vol. 8. KDU. 2015 (citado na pg. 11).

[Snellen *et al.* 2012] I Th M Snellen, Marcel Thaens e Wim BHJ van de Donk. *Public administration in the information age: Revisited*. Vol. 19. IOS press, 2012, pp. 53, 54 (citado na pg. 1).

[Wachsmuth *et al.* 2017] Henning Wachsmuth, Benno Stein e Yamen Ajjour. ““pagerank” for argument relevance”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. 2017, pp. 1117–1127 (citado na pg. 30).

[Yao *et al.* 2020] Tengjun Yao, Zhengang Zhai e Bingtao Gao. “Text classification model based on fasttext”. In: *2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS)*. IEEE. 2020, pp. 154–157 (citado nas pgs. 10, 11).