

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

USP Eventos

***Promovendo o Engajamento Estudantil na
Universidade***

Bruno Mazetti Saito, Willian Hiroshi Takihi

MONOGRAFIA FINAL

**MAC 499 — TRABALHO DE
FORMATURA SUPERVISIONADO**

Supervisor: Profº. Flávio Soares Correa da Silva

São Paulo
2023

*O conteúdo deste trabalho é publicado sob a licença CC BY 4.0
(Creative Commons Attribution 4.0 International License)*

Agradecimentos

The world breaks everyone, and afterward, some are strong at the broken places.

— Ernest Hemingway

Gostaria de agradecer primeiramente aos meus pais, Augusto e Denise, e à minha irmã, Tatyane, por sempre me apoiarem durante toda a vida e principalmente nos momentos mais difíceis, dentro e fora da graduação. Também gostaria de mencionar os amigos que fiz durante o curso e que me ajudaram diversas vezes e que compartilharam vários momentos marcantes dentro da graduação, e principalmente meu colega de trabalho Willian Hiroshi.

Bruno Mazetti Saito

Queria expor a minha gratidão a todos que me ajudaram a traçar meu caminho até a conclusão deste trabalho. Principalmente os meus pais, Carlos e Sonia, e familiares que foram a fonte constante de apoio e encorajamento não só durante a graduação, mas na vida. Aos grandes amigos que conheci na faculdade, Davi Menezes, Marcos Siolin, Luciano Leão e Lucas Harada, pelas experiências incríveis durante a jornada acadêmica. E finalmente, ao meu colega de equipe e amigo Bruno Mazetti, ajudando nas partes mais difíceis da faculdade.

Willian Hiroshi Takihi

Resumo

Bruno Mazetti Saito, Willian Hiroshi Takihi. **USP Eventos: Promovendo o Engajamento**

Estudantil na Universidade. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Em uma universidade como a USP, existem várias atividades e eventos que ocorrem diariamente, mas às vezes elas acabam passando despercebidas ou não há um modo fácil de filtrar aquelas que gostaríamos de participar. Portanto, o objetivo deste trabalho é criar um sistema capaz de compilar eventos universitários publicados em redes sociais, e mostrá-los em uma plataforma *web* acessível aos alunos. O esforço foi dividido em três diferentes frentes para a construção do mesmo: módulo de extração e processamento textual de postagens em redes sociais, *front-end* desenvolvido no *framework* React.JS para visualização de eventos e *back-end* utilizando o *framework* Django para gerenciamento de dados. O sistema final cumpre o que foi idealizado, permitindo que os alunos tenham mais facilidade para visualizar eventos de seu interesse dentro da faculdade.

Palavras-chave: Desenvolvimento *web*. Webscraping. Eventos.

Abstract

Bruno Mazetti Saito, Willian Hiroshi Takihi. **USP Events: Promoting Student Engagement on University.** Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

In a university like USP, there are various activities and events happening daily, however, sometimes they go unnoticed or there is no easy way to filter those we would like to participate in. Therefore, the goal of this project is to create a system capable of compiling university events published on social networks and displaying them on a web platform accessible to students. The effort during the development was divided into three different sections for its construction: a module for extracting and processing textual content from social media posts, a front-end developed in the React.JS framework for event visualization, and a back-end using the Django framework for data management. The final system fulfills what was envisioned, allowing students to easily view events of interest within the university.

Keywords: Web development. Webscraping. Events.

Listas de Abreviações

API <i>Application Programming Interface</i>	3
HTTP <i>Hypertext Transfer Protocol</i>	4
IA Inteligência Artificial	14
RDBMS <i>Relational Database Management System</i>	4
REST <i>Representational State Transfer</i>	3
UI <i>User Interface</i>	5
USP Universidade de São Paulo	1
UX <i>User Experience</i>	5
URL <i>Uniform Resource Locator</i>	13

Lista de figuras

3.1	Quadro KanBan com as tarefas na plataforma Trello	8
4.1	Exemplo de Teste unitário (1)	10
4.2	Exemplo de Teste unitário (2)	10
4.3	Modelo do banco de dados utilizado	10
5.1	Página principal do sistema	13
5.2	Detalhes de um evento	14
5.3	Filtros de eventos	14
5.4	Processamento de um possível evento	15
5.5	Processamento textual no servidor	16
5.6	Processamento textual via <i>Huggingface chatbot</i>	16
5.7	Processamento textual de um não evento pelo <i>Mistral 7B</i>	17
5.8	Processamento textual de um não evento pelo <i>Llama-2</i>	17
5.9	Instruções para extração de informações do evento	17
5.10	Página de eventos sem localização	19
5.11	Eventos duplicados	20
5.12	Botão para abrir janela de cadastro ou <i>login</i>	20
5.13	Janela de <i>login</i> de usuários	21
5.14	Janela de cadastro de novo usuário	21
5.15	Botão de acesso à área do usuário	21
5.16	Formulário de registro de <i>webpages</i>	22
5.17	Aparência da página com <i>webpages</i> cadastradas	22
5.18	Ícones de ação de <i>webpages</i>	23

Sumário

Lista de Abreviações	vii
1 Introdução	1
2 Tecnologias	3
2.1 <i>Back-end</i>	3
2.1.1 Django	4
2.1.2 SQLite	4
2.2 <i>Front-end</i>	4
2.2.1 React.JS	5
2.3 <i>Webscraping</i>	5
2.3.1 Selenium e BeautifulSoup	5
3 Metodologia	7
4 Arquitetura	9
4.1 <i>Clean Code</i>	9
4.2 Banco de Dados	10
4.2.1 <i>User</i>	11
4.2.2 <i>WebPage</i>	11
4.2.3 <i>Event</i>	11
4.2.4 <i>WebPageUsers</i>	12
5 Fluxo da plataforma	13
5.1 Amostragem de eventos	13
5.1.1 Raspagem de dados	15
5.1.2 Processamento de informações	16
5.1.3 Código aberto	18
5.2 Eventos sem localização	18

5.3	Controle de usuários	20
5.4	Cadastro de páginas	21
5.4.1	Controle de páginas para usuários diferentes	23
6	Pontos de melhoria	25
6.1	Cadastro de eventos pelos usuários	25
6.2	Sistema em contêiner Docker	26
7	Conclusão	27
	Referências	29

Capítulo 1

Introdução

A diversidade de eventos e atividades disponíveis em uma universidade de renome como a Universidade de São Paulo ([USP](#)) é notável. Desde palestras e conferências de destaque até eventos esportivos, culturais e acadêmicos, os estudantes têm acesso a uma infinidade de oportunidades para aprimorar seus conhecimentos e se envolver em atividades enriquecedoras.

A participação ativa dos alunos em eventos acadêmicos e extracurriculares é fundamental para enriquecer sua experiência universitária e contribuir para seu desenvolvimento pessoal e profissional. Como ressaltado por [TINTO \(2012\)](#) em *Leaving College: Rethinking the Causes and Cures of Student Attrition* a participação em atividades fora da sala de aula está diretamente relacionada à retenção estudantil e ao sucesso acadêmico. Além disso, estudos como o de [ASTIN et al. \(1997\)](#) em *What Matters in College: Four Critical Years Revisited* destacam que o envolvimento dos alunos em atividades extracurriculares está positivamente associado ao desenvolvimento de habilidades de liderança, resolução de problemas e ao aumento da satisfação com a experiência universitária.

Atualmente as redes sociais têm um papel fundamental com relação ao acesso à informação, já que uma grande diversidade de sites e páginas podem ser facilmente acessados através de um *smartphone* ou computador. Porém, justamente pelo fato de existirem tantas fontes de informação, novos alunos tendem a ficar perdidos frente a um mar de oportunidades tão vasto e com tantas opções de atividades extracurriculares disponíveis. Assim, o encontro de informações e divulgações de eventos que realmente sejam interessantes para o aluno, pode se tornar um tanto difícil.

Neste contexto, surge a necessidade de um sistema que possa simplificar o acesso às informações relacionadas a eventos e atividades que estejam próximas de ocorrer nos arredores da universidade. Uma plataforma simples e intuitiva que seja capaz de juntar essas informações, a partir de diferentes publicações de redes sociais, e que permita que seus usuários sejam capazes de visualizar, de forma clara e concisa, informações sobre localização, horário e preço dos eventos que sejam de seu interesse.

Com esse problema em vista, este Trabalho de Conclusão de Curso aborda a criação de um sistema que seja capaz de atender esta demanda e, consequentemente, facilitar e incentivar a participação de alunos na vida universitária. É importante ressaltar que este

projeto tem como foco, inicialmente, apenas o campus da Cidade Universitária da USP, mas que possa ser facilmente escalável para outras universidades e até mesmo regiões específicas.

Capítulo 2

Tecnologias

O intuito inicial do projeto era criar uma plataforma que pudesse ser facilmente acessada através de navegadores que todos estão acostumados a usar no dia-a-dia, dessa forma, tornou-se necessário a definição de como poderia ser estruturado o projeto e quais as ferramentas a serem utilizadas a fim de viabilizar a criação de uma plataforma *web* que atendessem ao objetivo inicial do projeto.

Em relação à estrutura do projeto, a princípio, a separação mais clara e óbvia foi a escolhida. O sistema foi dividido em dois sub projetos, *back-end* o qual ficaria responsável pelo armazenamento de dados, obtenção dos textos fonte e processamento das informações; e *front-end* que seria utilizado para exibição dos dados e interação do usuário com a plataforma.

Ao longo do desenvolvimento foi se tornando cada vez mais clara uma nova separação para a estrutura geral, assim, um novo módulo chamado de *scripts* foi criado no projeto, onde foram alocados arquivos que não se encaixavam exatamente no escopo que estava sendo definido para o *back-end*.

Dessa forma, a aplicação está construída em três partes principais: *front-end*, *back-end* e *webscraper scripts*. A seguir, será descrita em mais detalhes a escolha das tecnologias utilizadas e o objetivo de cada camada.

2.1 *Back-end*

O *back-end* é uma parte crucial de um sistema *web* responsável pelo processamento, gerenciamento, armazenamento e segurança de dados. Ele age como uma camada intermediária que permite aos usuários interagirem com o sistema sem a necessidade de entender as complexidades de suas operações.

A comunicação com o *front-end* foi feita implementando a arquitetura *Representational State Transfer (REST)* e *Application Programming Interfaces (APIs)*. Essa abordagem permite ao *front-end* solicitar dados e realizar ações no *back-end* de maneira eficiente e padronizada, tornando-se amplamente utilizada em sistemas *web*.

No projeto, optou-se pelo *framework* Django como ponto de partida para a criação da API.

2.1.1 Django

Uma das partes principais do projeto é o conjunto de *scripts* para automação da coleta e processamento de dados de redes sociais escritos em Python. Já que o Django é também baseado nessa linguagem, isso permite uma interação direta e simples entre ambas as camadas (*back-end* e *scripts*).

Dado que nunca tivemos contato com o Django, um fator decisivo para sua escolha é a sua grande comunidade de usuários, o que proporciona muito conhecimento online, uma documentação detalhada e pronta ajuda em eventuais problemas e dúvidas. Ainda mais, ele possui código aberto, alinhando-se perfeitamente com a natureza acadêmica do projeto.

Outro ponto que acelerou significativamente o desenvolvimento foi a vasta quantidade de bibliotecas disponíveis para o Django. A autenticação de usuários dentro da plataforma é um dos exemplos que foi facilitada pelo uso de bibliotecas.

2.1.2 SQLite

Dado que o sistema necessita gerenciar dados sobre essencialmente usuários e possíveis eventos futuros, estava claro o uso de um banco de dados relacional desde o início. Essas entidades possuem um formato predefinido e podem ser facilmente conectados em um modelo de dados relacional, portanto restava apenas decidir qual programa utilizar para manejá-la informação.

O SQLite é um *Relational Database Management System (RDBMS)* de código aberto responsável por criar, atualizar e gerenciar bancos de dados relacionais. Uma de suas principais características é o armazenamento de informações em arquivos locais, dispensando o uso de um servidor de banco de dados separado. Essa propriedade torna o SQLite adequado para projetos de menor escala como o construído, uma vez que não exige a complexidade de um processo diferente com o servidor em execução, como o PostgreSQL ou MySQL.

Outro ponto de interesse é sua capacidade de manejá-la baixo a médio tráfego de requisições *HTTP*, o que é ideal para o caso de uso, já que inicialmente, não existirão muitos usuários na plataforma.

Além disso, o SQLite já é integrado nativamente ao *framework* Django. Isso permitiu que o desenvolvimento fosse mais ágil, sem necessidade de configurações adicionais para ter acesso ao banco de dados.

2.2 *Front-end*

O *front-end* é igualmente importante em um sistema *web*, responsável pela interface gráfica disponibilizada ao usuário final. Dois aspectos englobados pelo *front-end* são: expe-

riência do usuário, a apresentação de informações e a interatividade, conjunto denominado de *User Experience* ([UX](#)); e a experiência visual do sistema voltado a estética da plataforma, área denominada de *User Interface* ([UI](#)).

Existem inúmeros *frameworks* que oferecem ferramentas para o desenvolvimento dessas camadas do *front-end* de maneira mais simples e ágil. Alguns deles são Angular, Svelte, Vue.JS e React.JS. A seguir, serão listados alguns motivos pelos quais decidiu-se pelo uso deste último.

2.2.1 React.JS

O principal motivo da escolha do React.JS para a construção do *front-end* foi a sua enorme popularidade, o que assegurou a existência de muito conteúdo *online* e auxílio da comunidade disponível. Isso também indiretamente deu acesso a muitas bibliotecas externas como o *Material-UI* que oferece componentes de [UI](#) prontos para uso, permitindo o foco nas funcionalidades do projeto.

Além disso, ambos os integrantes do time já trabalharam com React.JS em projetos passados dentro da universidade, como na matéria de MAC0472 (Laboratório de Métodos Ágeis). Essa oportunidade proporcionou uma base robusta e confortável para a utilização deste mesmo *framework* neste trabalho, reduzindo a curva de aprendizado e aumentando a qualidade geral do projeto.

2.3 Webscraping

A plataforma necessita de informações de possíveis eventos oriundas de fontes externas. A coleta desses dados é automatizada utilizando *scripts* escritos em Python e realizada em cima de redes sociais usando ferramentas como Selenium e BeautifulSoup, essa técnica para adquirir os dados é denominado de *Webscraping* ou raspagem de dados.

2.3.1 Selenium e BeautifulSoup

Existem várias bibliotecas mantidas em Python específicas para raspagem de informações de redes sociais, facilitando bastante o processo. Porém, é comum o bloqueio da utilização dessas ferramentas, por conta de possíveis violações dos termos de uso de cada plataforma. Por esse motivo, optou-se pelo uso do Selenium, comumente empregado em testes automatizados do *front-end*, mas podendo ser aplicado também para *Webscraping*.

Selenium é uma ferramenta capaz de automatizar funções do navegador, possibilitando interagir com *websites* como se fossem usuários comuns, realizando ações como cliques e deslize de tela. Por conta dessa natureza, o problema de suspensão durante a navegação *web* é menos frequente e oferece mais flexibilidade ao procurar informações nas redes sociais.

Enquanto o Selenium busca o conteúdo, o BeautifulSoup é responsável por efetivamente extrair as informações. Ele é uma biblioteca capaz de obter dados de arquivos HTML,

permitindo adquirir o texto de *tags* HTML específicas, sendo exatamente o caso de uso do sistema para obter o conteúdo das postagens das mídias sociais.

Capítulo 3

Metodologia

Neste capítulo, será descrito a visão geral a respeito do método de trabalho que foi aplicado durante a elaboração do sistema e de como as tarefas a serem desenvolvidas foram organizadas entre os integrantes do grupo.

Mesmo antes que a escolha em relação ao tema que seria desenvolvido durante o ano fosse decidida, já havia um consenso de que a organização do trabalho a ser aplicada se basearia na metodologia ágil.

A metodologia ágil é um conceito amplamente usado no mundo do desenvolvimento de software, que utiliza o manifesto ágil *Agile Manifesto* (2001) como base e que, resumidamente, é constituído de um conjunto de práticas que visa uma entrega contínua ao longo do projeto, que possibilite estruturar uma relação de tarefas clara e intuitiva durante todo o processo, e que seja capaz de manter todos os membros envolvidos alinhados quanto ao *status* do desenvolvimento.

A escolha pela metodologia ágil se deu pelo fato de que ambos os integrantes já haviam tido contato com esse método de trabalho, sendo o primeiro deles a partir da disciplina MAC0472 (Laboratório de Métodos Ágeis) e também utilizado no desenvolvimento do projeto principal na disciplina MAC0413 (Tópicos Avançados de Programação Orientada a Objetos).

Um dos tipos mais aplicados do conceito de metodologia ágil é o chamado *KanBan*. Como descrito por WAKODE *et al.* (2015), esta prática se baseia no uso de um quadro que é preenchido com diversos cartões que descrevem como está o andamento do projeto. Neles ficam descritas atividades que deveriam ser realizadas, destacando entre elas quais têm a maior prioridade; relatam quais pontos estão sendo desenvolvidos no momento; além de também manterem registros dos pontos que já foram entregues e, portanto, já estão implementados no serviço.

Este trabalho usou o sistema do *KanBan* como modelo para guiar o processo de desenvolvimento. Para isto, foi utilizada a plataforma *Trello* onde, assim como descrito no último parágrafo, os *status* das tarefas foram mantidos e registrados ao longo do desenvolvimento. O quadro com as tarefas pode ser acessado através do endereço <https://trello.com/tcc-eventos-usp>.

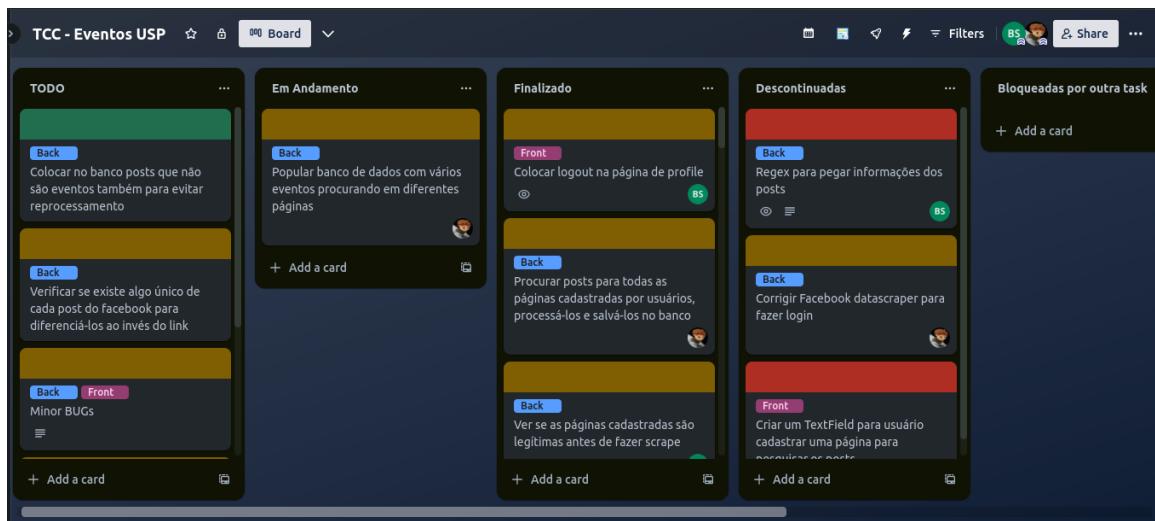


Figura 3.1: Quadro KanBan com as tarefas na plataforma Trello

As tarefas a serem desenvolvidas foram separadas de acordo com seu escopo dentro da divisão macro do sistema, isto é, eram classificadas como sendo direcionadas ao *back-end* e ao *front-end*. Além disso, foi feita uma classificação com relação à dificuldade de execução das tarefas a partir de uma estimativa prévia, sendo essas consideradas fáceis (verde), médio (amarelo) e difícil (vermelho).

Capítulo 4

Arquitetura

Neste capítulo, os pontos a respeito da organização e estruturação da arquitetura que moldaram o formato do projeto, serão descritos com alguns detalhes, primeiramente, destacando alguns tópicos adotados referentes ao conceito de código limpo e por fim detalhando a modelagem do banco de dados e a explicação da construção de suas entidades e relações.

4.1 *Clean Code*

Um bom projeto não se resume apenas em eficiência na execução do programa ou em um visual atraente para o usuário, outro ponto de grande importância, que muitas vezes auxilia no alcance das qualidades citadas, é uma boa organização de como os arquivos vão ser estruturados e separados por escopo.

Como descrito no capítulo 2 deste documento, houve uma separação base da estrutura do projeto em três diferentes frentes. Além deste formato, durante o desenvolvimento do trabalho, a arquitetura dos arquivos dentro destes tópicos macro sempre foi um tópico recorrente, de forma que fosse possível manter uma boa qualidade sobre o código que estava sendo escrito, ponto este, que foi constantemente abordado durante toda a graduação ao longo de disciplinas como MAC0218 (Técnicas de Programação II).

Assim como relatado por MARTIN (2008) em *Clean Code: A Handbook of Agile Software Craftsmanship*, um código pode ser considerado limpo se é compreendido facilmente por desenvolvedores que não necessariamente participaram do processo de desenvolvimento, o que torna muito mais viável uma possível manutenção ou melhoria futura no código. Este ideal consiste de um conjunto de conceitos básicos que em conjunto melhoram o entendimento do que é escrito.

Nomes de variáveis e funções autoexplicativos, métodos enxutos e que possuam uma responsabilidade única e bem definida, trechos de código relacionados entre si próximos, consistência de códigos similares em lugares diferentes, aplicação de padrões de projetos arquiteturais, testes unitários e uso de estruturas de dados foram alguns dos diversos tópicos que foram utilizados no desenvolvimento e que corroboraram para a caracterização de um código limpo.

```

13 class TestGetProcessPosts(unittest.TestCase):
14     @mock.patch.object(APIRequester, "get_all_events")
15     @mock.patch("getProcessPosts._get_all_posts")
16     @mock.patch("getProcessPosts.process_post")
17     @mock.patch("getProcessPosts.get_lat_lon_by_address")
18     @mock.patch("getProcessPosts.logger")
19     def test_get_process_posts_correctly_if_link_not_on_database(
20         self,
21         mock_logger,
22         mock_get_lat_lon_by_address,
23         mock_process_post,
24         mock_get_all_posts,
25         mock_get_all_events,
26     ):
27         # mocks
28         mock_get_all_events.return_value = []
29
30         mock_webpage = WebPage(
31             id=1,
32             link="http://mock.link",
33             source="Instagram",
34             users=[],
35         )
36
37         mock_raw_post = RawPost(
38             post_text="Text",
39             post_link="http://mock.link",
40             post_source="Instagram",
41             webpage=mock_webpage,
42         )
43         mock_get_all_posts.return_value = [mock_raw_post]

```

Figura 4.1: Exemplo de Teste unitário (1)

```

44     mock_processed_post = {
45         "address": "Mock address",
46         "date": "01/01/2023",
47         "price": 0,
48         "type": "unclassified",
49     }
50     mock_process_post.return_value = mock_processed_post
51
52     mock_get_lat_lon_by_address.return_value = ("0", "0")
53
54     # call function
55     result = get_process_posts()
56
57     # asserts
58     expected_result = [
59         {
60             "address": "Mock address",
61             "date": "01/01/2023",
62             "price": 0,
63             "type": "unclassified",
64             "lat": "0",
65             "lng": "0",
66             "post_link": "http://mock.link",
67             "source": "Instagram",
68             "webpage": 1,
69         }
70     ]
71
72     self.assertEqual(result, expected_result)
73     mock_logger.error.assert_not_called()
74

```

Figura 4.2: Exemplo de Teste unitário (2)

4.2 Banco de Dados

O banco de dados foi modelado de uma maneira que pudesse se manter simples, mas ao mesmo tempo fosse capaz de atender as necessidades para a estruturação do projeto, para isso, algumas disciplinas cursadas durante a graduação como MAC0350 (Introdução ao Desenvolvimento de Sistemas de Software) e MAC0426 (Sistemas de Bancos de Dados) contribuíram com os conceitos básicos utilizados.

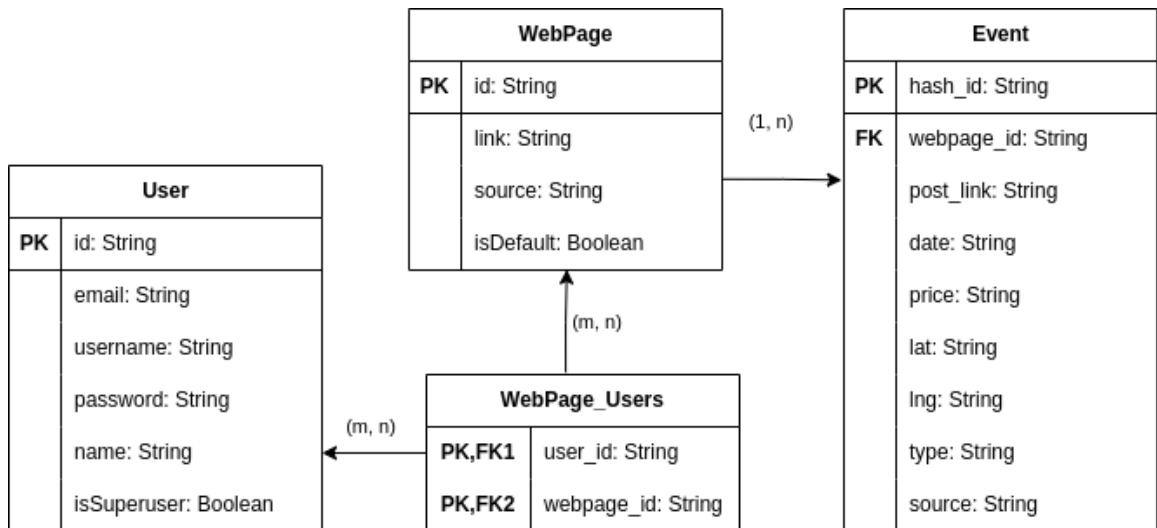


Figura 4.3: Modelo do banco de dados utilizado

Existem três principais entidades definidas no modelo. Para cada uma, existem APIs responsáveis pela criação, deleção ou edição delas. Algumas destas APIs necessitam de privilégios de administradores para serem utilizadas. A seguir estão alguns detalhes com respeito às entidades e tabelas modeladas:

4.2.1 *User*

A tabela de *User* armazena informações a respeito dos usuários da plataforma. Um novo usuário é criado toda vez que a ação de cadastro é feita dentro da plataforma a partir de informações básicas como nome de usuário, senha e *email* por exemplo. O armazenamento da senha criada no registro é armazenada de forma criptografada dentro do banco para garantir uma maior segurança dessa informação.

Existem dois tipos de usuários, os criados através do fluxo descrito são usuários padrões, enquanto que usuários criados utilizando a [API](#) direcionada à uso exclusivo de administradores são chamados de **super usuários**. Os tipos de usuários podem ser diferenciados no banco de dados através do campo *isSuperuser* da tabela *User*.

4.2.2 *WebPage*

A tabela *WebPage* é utilizada para guardar informações relacionadas às páginas de redes sociais de preferência cadastradas pelos usuários dentro da plataforma. O sistema utiliza destas *webpages* cadastradas para buscar as informações necessárias para criação do evento que será exibido dentro da plataforma futuramente.

Como descrito no parágrafo anterior, as *webpages* são o ponto inicial para que um evento seja gerado, porém, existem situações em que um usuário deseja utilizar a plataforma sem a necessidade de realizar um cadastro, consequentemente, não haverá nenhuma *webpage* de preferência criada por ele. Dessa forma, inicialmente o sistema possui *webpages* padrões de onde serão extraídos os eventos mostrados para todos os usuários, com e sem cadastro na plataforma.

É importante ressaltar que atualmente o sistema dá suporte para páginas que sejam do *Facebook* ou do *Instagram*, por isso, a própria plataforma restringe a criação das *webpages* de uma destas duas fontes, representado pelo campo *source* na tabela.

4.2.3 *Event*

A tabela *Event* é responsável pelo armazenamento dos eventos gerados a partir das *webpages*. Os textos que são obtidos dos *posts* publicados nessas páginas passam por uma etapa de processamento onde são extraídas as informações necessárias para a construção dos eventos. Dessa forma, é estabelecida uma relação entre as entidades *Event* e *Webpage* onde uma *webpage* pode possuir um ou mais eventos diferentes.

A princípio, a partir do processamento os eventos são diferenciados em dois tipos que são esportivos (*sport*) ou cultura (*culture*), entretanto, existem casos que o modelo não consegue classificar corretamente o evento em um desses tipos, assim, estes são mantidos como não classificados (*unclassified*). Os diferentes tipos de eventos são identificados dentro do sistema através do campo *type* na tabela *Event*. Essa entidade ainda possui um campo de *id* que utiliza um sistema de *hash* que utiliza o *link* do post com os dados do evento para a geração desse *id*.

4.2.4 WebPageUsers

Essa tabela foi criada para funcionar como um intermediário entre as entidades de *Event* e *WebPage*. Essa relação tem o intuito de tornar possível atrelar um usuário aos eventos obtidos através de uma *webpage* cadastrado pelo usuário. Além disso, como a relação entre estas entidades mencionadas foi estabelecida, também é possível criar uma relação para identificar quais usuários realizaram o cadastro de cada uma das *webpage* existentes.

Capítulo 5

Fluxo da plataforma

Neste capítulo, será explorado o funcionamento da plataforma como um todo, informando o que foi necessário ser estruturado para alcançar o estágio final de desenvolvimento e os respectivos desafios de projeto e técnicos encontrados.

5.1 Amostragem de eventos

Antes de tudo, a plataforma pode ser acessada via URL pública <https://uspevents.ix.tc/>. A primeira página acessada pelo usuário é onde são amostrados os eventos que vão ocorrer ao longo do próximo mês.



Figura 5.1: Página principal do sistema

Nela, é possível clicar na localização de eventos disponíveis no mapa. Isso dá acesso às informações mais granulares sobre eles, como a localização, a data do evento, o preço de entrada caso exista e um *link* para a sua postagem original.

No canto superior direito da página, o usuário pode filtrar o período no qual deseja ver eventos disponíveis e o tipo deles, como eventos culturais e esportivos.

Figura 5.2: Detalhes de um evento

O primeiro passo necessário para criação do sistema e alimentação desta página com os dados necessários foi o processo de *webscraping* ou raspagem de dados. Como foi brevemente descrito no capítulo 2, a plataforma obtém dados de eventos a partir da aquisição e processamento de textos encontrados nas postagens em redes sociais.

Para tanto, criou-se um *script* executado a cada 6 horas que busca em páginas do *facebook* e *instagram* pelos textos das últimas 5 postagens de cada uma delas. Em seguida, eles são encaminhados para uma ferramenta oferecida pelo website *Hugginface chatbot*, no qual solicita-se que o texto seja processado em busca das informações necessárias para estruturar um possível evento dentro da plataforma.

O *Hugginface chatbot* é basicamente uma plataforma que te permite interagir com diferentes modelos de Inteligência Artificial (*IA*) generativa capazes de responder perguntas gerais ou executar tarefas requisitadas. Eles foram treinados para entender e gerar textos semelhantes a seres humanos.

Finalizado o processamento textual pelo *chatbot*, o próximo passo é encontrar a geoloca-

5.1 | AMOSTRAGEM DE EVENTOS

O Centro Universitário Maria Antonia inaugura, no dia 27 de outubro, às 19h, a exposição Batalha a preservar: Maria Antônia, 55 anos – fotografias Hiroto Yoshioka. Vinculada às comemorações dos 30 anos de inauguração do Centro MariAntonia, a entrada é gratuita! Venha conferir!#mariantoniusp #exposição #batalhadamariaantonia

Colete informações sobre o possível evento acima: DATA de quando o evento acontecerá no formato string yyyy-mm-dd (use a seguinte data 2023-10 como referência). ENDEREÇO de onde acontecerá o evento como string. PREÇO do evento como número, coloque 0 caso o evento seja gratuito. Classifique-o como evento cultura ("culture"), esporte ("sport") ou não classificado ("unclassified"). Informe apenas os dados em um formato JSON, sem nenhuma informação adicional. Os campos do JSON devem ser "date", "address", "price" e "type". Caso o texto acima não seja um evento, apenas retorne string "null".

```
{
  "date": "2023-10-27",
  "address": "Centro Universitário Maria Antonia",
  "price": 0,
  "type": "culture"
}
```

Figura 5.4: Processamento de um possível evento

lização com latitude e longitude do evento. Utilizou-se o serviço gratuito de geocodificação chamado Nominatim para isso, no qual é feita uma requisição via [API](#) de um endereço e a saída é a sua respectiva geolocalização.

Finalmente, caso a maioria das informações sejam encontradas, é possível criar um evento que é salvo no banco de dados para consumo pelos usuários da plataforma.

Durante esse processo, surgiram diversos empecilhos e decisões de projeto pertinentes que serão discutidos a seguir.

5.1.1 Raspagem de dados

Como mencionado no capítulo 2, um dos grandes problemas de realizar a raspagem de dados via alguma biblioteca externa em mídias sociais é a suspensão temporária na utilização da aplicação devido a alguma infração nos termos de uso, mesmo que as páginas sejam abertas ao público.

Esse problema foi contornado utilizando a ferramenta Selenium para simular o uso por um usuário comum e salvar a sessão do navegador. Nesses sites, é comum que precise de uma conta para acessar o seu conteúdo, mas conectar-se toda vez que executar o processo de *webscraping* não é o ideal. Por isso que é interessante utilizar algum método para manter a sessão da conta salva para futuras raspagens de dados.

Um desafio adicional nesta etapa é referente às potenciais mudanças de design ou funcionalidades nos *websites*. Ao longo do desenvolvimento do projeto, diversas modificações foram implementadas nas suas estruturas HTML, deixando o processo de raspagem de dados desatualizado. Consequentemente, todo o processamento textual ficava comprometido, demandando um tempo considerável para identificação e correção dos erros. Infelizmente,

não existe um método simples para contornar esse tipo de problema, o que torna esse processo particularmente tedioso.

5.1.2 Processamento de informações

Alguns pontos importantes a se discutir para que o processamento textual fosse possível são: processo custoso e demorado, escolha do modelo de IA generativa; instruções para guiar o modelo na extração de informações.

A ideia inicial era que o processamento dos textos como descrito anteriormente fosse realizado em tempo de execução. O que significa que ele executaria a cada chamada de API que o usuário fizesse para requisitar possíveis eventos na plataforma. Isso logo foi considerado inviável, já que todo o procedimento é demorado e esse tempo escala à medida que mais textos são processados. A desvantagem considerada é o atraso de 6 horas de escrita no sistema caso uma nova postagem seja feita de um evento nas redes sociais.

O alto consumo computacional nesse processo também fez com que se optasse deixá-lo a encargo de um terceiro, no caso o *Huggingface chatbot*. O servidor utilizado para hospedar a plataforma dispõe apenas de 4 núcleos de uma CPU ARM de propósito geral, o que não é ideal para esse caso de uso. Nessas condições, utilizando o modelo de IA generativa **Llama-2** recentemente disponibilizado pela empresa Meta, foi possível processar um texto de um possível evento em aproximadamente 15 minutos e meio. Conforme mais postagens são analisadas, o tempo gasto se torna absurdo em um cenário normal de uso da plataforma, enquanto o *chatbot* utilizado é consideravelmente mais rápido. Porém, essa escolha torna essa etapa dependente de um serviço externo, o que dificulta a manutenção já que podem ocorrer quebras de contrato inesperadas ou indisponibilidade do mesmo por certo período, impactando diretamente nas operações do projeto.

```
[02:43:34 ON dataparser.py:31] INFO: Processando o texto: E amanhã, quinta, 26 de outubro, a partir das 16 horas, o Sarau Comungar se apresenta no Centro Universitário Maria Antonia, dentro da programação paralela da exposição "São Mateus move o centro", que fica em cartaz até 28 de janeiro de 2024. Venha participar! Entrada gratuita!
[02:43:34 ON dataparser.py:39] INFO: Processamento durou 15.61 minutos
[02:43:34 ON dataparser.py:49] INFO: Resultado: {'date': '2023-11-26', 'address': 'Centro Universitário Maria Antonia', 'price': 0, 'type': 'culture'}
```

Figura 5.5: Processamento textual no servidor



Figura 5.6: Processamento textual via Huggingface chatbot

5.1 | AMOSTRAGEM DE EVENTOS

O *chatbot* permite interagir com 5 modelos de código aberto diferentes até o momento. Dentre eles, está o anteriormente mencionado **Llama-2**, o recente **OpenChat 3.5** e o utilizado **Mistral 7B**. A decisão da escolha do modelo foi feita a partir da qualidade geral das informações extraídas. No final, observou-se que o *Mistral 7B* apresentou maior acurácia ao executar a instrução fornecida em comparação com os demais, principalmente na tarefa de julgar se o texto processado corresponde a um possível evento ou não.

```
São 110.399 candidatos inscritos que concorrem a vagas em diversos cursos da USP. Medicina em São Paulo, Ribeirão Preto e Bauru lideram ranking de concorrência, seguidos de Psicologia e Relações Internacionais. Os candidatos inscritos no próximo vestibular da Fvest podem consultar a relação de candidatos por vaga por curso na concorrência para ingresso na USP. No ranking das carreiras, o curso de Medicina continua o mais concorrido nos três campi em que é oferecido. Em São Paulo, ficou em primeiro lugar com 117,7 candidatos por vaga, seguido de Medicina em Ribeirão Preto, com 86,6, e Medicina em Bauru, com 78,2. Psicologia, em São Paulo, e Relações Internacionais são as mais concorridas na sequência. O número de inscritos é de 99.573 candidatos e 10.826 treineiros, estudantes que ainda não concluíram o ensino médio, totalizando 110.399 pessoas. Em 2023, o número total de inscritos foi de 114.434. Confira a relação completa no link da bio.
```

Colete informações sobre o possível evento acima: DATA de quando o evento acontecerá no formato string yyyy-mm-dd (use a seguinte data 2023-11 como referência). ENDEREÇO de onde acontecerá o evento como string. PREÇO do evento como número, coloque 0 caso o evento seja gratuito. Classifique-o como evento cultura ("culture"), esporte ("sport") ou não classificado ("unclassified"). Informe apenas os dados em um formato JSON, sem nenhuma informação adicional. Os campos do JSON devem ser "date", "address", "price" e "type". Caso o texto acima não seja um evento, apenas retorne string "null".

• null

Figura 5.7: Processamento textual de um não evento pelo *Mistral 7B*

```
São 110.399 candidatos inscritos que concorrem a vagas em diversos cursos da USP. Medicina em São Paulo, Ribeirão Preto e Bauru lideram ranking de concorrência, seguidos de Psicologia e Relações Internacionais. Os candidatos inscritos no próximo vestibular da Fvest podem consultar a relação de candidatos por vaga por curso na concorrência para ingresso na USP. No ranking das carreiras, o curso de Medicina continua o mais concorrido nos três campi em que é oferecido. Em São Paulo, ficou em primeiro lugar com 117,7 candidatos por vaga, seguido de Medicina em Ribeirão Preto, com 86,6, e Medicina em Bauru, com 78,2. Psicologia, em São Paulo, e Relações Internacionais são as mais concorridas na sequência. O número de inscritos é de 99.573 candidatos e 10.826 treineiros, estudantes que ainda não concluíram o ensino médio, totalizando 110.399 pessoas. Em 2023, o número total de inscritos foi de 114.434. Confira a relação completa no link da bio.
```

Colete informações sobre o possível evento acima: DATA de quando o evento acontecerá no formato string yyyy-mm-dd (use a seguinte data 2023-11 como referência). ENDEREÇO de onde acontecerá o evento como string. PREÇO do evento como número, coloque 0 caso o evento seja gratuito. Classifique-o como evento cultura ("culture"), esporte ("sport") ou não classificado ("unclassified"). Informe apenas os dados em um formato JSON, sem nenhuma informação adicional. Os campos do JSON devem ser "date", "address", "price" e "type". Caso o texto acima não seja um evento, apenas retorne string "null".

{
 "date": "2023-11-01",
 "address": "Universidade de São Paulo",
 "price": 0,
 "type": "unclassified"
}

Figura 5.8: Processamento textual de um não evento pelo *Llama-2*

É importante ressaltar que esses modelos de **IA** generativa foram utilizados com configurações padrão, sem quaisquer mudanças em possíveis parâmetros que pudessem melhorar a qualidade do resultado da tarefa, dado que o *Huggingface chatbot* não oferece essa opção e essa etapa não é o foco do projeto em si, mas sim a construção da plataforma de ponta a ponta.

Além da escolha do modelo, o modo como a instrução é estruturada para que ele execute a tarefa também interfere diretamente na qualidade dos resultados obtidos. O processo de construção eficaz dessas instruções direcionadas para que a **IA** retorne um resultado mais preciso está sendo recentemente denominada de Engenharia de *Prompts*.

```
Colete informações sobre o possível evento acima: DATA de quando o evento acontecerá no formato string yyyy-mm-dd (use a seguinte data 2023-11 como referência). ENDEREÇO de onde acontecerá o evento como string. PREÇO do evento como número, coloque 0 caso o evento seja gratuito. Classifique-o como evento cultura ("culture"), esporte ("sport") ou não classificado ("unclassified"). Informe apenas os dados em um formato JSON, sem nenhuma informação adicional. Os campos do JSON devem ser "date", "address", "price" e "type". Caso o texto acima não seja um evento, apenas retorne string "null".
```

Figura 5.9: Instruções para extração de informações do evento

O texto acima corresponde às instruções ou *prompt* utilizado para extração das informações desejadas. Algumas boas práticas compiladas por **BIGELOW, 2023** foram seguidas

no ramo de Engenharia de *Prompts*:

- **Especificidade:** Ao definir exatamente quais informações devem ser buscadas, no caso a data, o endereço, o preço e o tipo de evento, é possível restringir a gama de possíveis respostas e aumentar a probabilidade do modelo produzir a saída desejada.
- **Uso de exemplos:** É provável que o texto não apresente uma data completa do evento, como por exemplo "*No dia 08, a apresentação artística acontecerá...*". Fornecendo uma data de referência pode ajudar o modelo a entregar uma data inteira que se espera na saída. O mesmo serve para os demais campos, em que se informa o formato do tipo do evento e preço.
- **Formato de saída claramente definido:** Especificar que a informação deve ser retornada em formato JSON foi essencial para que se pudesse interpretá-lo programaticamente posteriormente.

Porém, é extremamente importante destacar que o modelo ainda comete muito mais erros do que acertos, portanto as informações disponibilizadas dentro da plataforma devem ser usadas com cautela.

5.1.3 Código aberto

A natureza acadêmica da plataforma e a escassez de recursos monetários fizeram com que fosse claro a preferência pelo uso de tecnologias de código aberto. Essa escolha, embora alinhada com os princípios de transparência e colaboração da comunidade acadêmica, não esteve isenta de desafios durante o desenvolvimento do projeto.

Inicialmente, o processo de geocodificação necessário para adquirir a geolocalização dos eventos foi feito por meio da [API](#) do Google Maps, por conta de sua velocidade e principalmente sua precisão. O Nominatim, por ser uma solução de código aberto baseada em dados do OpenStreetMap, apresentou limitações em termos de acurácia na maioria dos casos.

A principal questão que surgiu foi a diferença na abrangência dos dados geoespaciais entre as duas opções. Enquanto o Google Maps possui uma extensa base de dados, alimentada por fontes diversas e frequentemente atualizada, o Nominatim depende das contribuições da comunidade OpenStreetMap, o que pode resultar em informações menos abrangentes e atualizadas.

5.2 Eventos sem localização

Uma das informações mais importantes contidas em um evento é a sua localização ou endereço, haja vista que é a partir dela que é possível realizar a busca da geolocalização e, em consequência, exibir o local de acontecimento do evento no mapa.

Entretanto, durante a etapa de processamento dos eventos, assim como destacamos no tópico [Processamento de informações](#), existem momentos em que o modelo não é capaz de obter corretamente as coordenadas referentes ao evento, seja por um erro do próprio

modelo em identificar o endereço, ou por uma busca incorreta das coordenadas correspondentes mesmo que o dado tenha sido encontrado corretamente, ou até mesmo porque o texto analisado não contém essa informação de maneira explícita. Independentemente do motivo da falha, neste cenário o sistema não consegue exibir na página principal estes eventos processados.

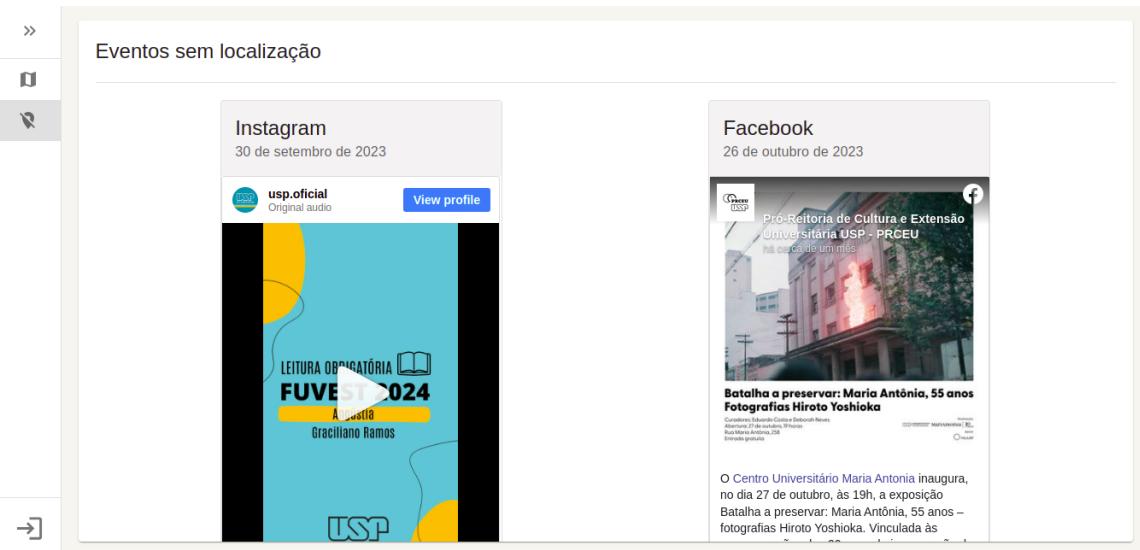


Figura 5.10: Página de eventos sem localização

Por este motivo, a página de eventos de localização foi idealizada com o intuito de reunir os eventos dentro do sistema mesmo com a menor quantidade possível de informações obtidas a partir do modelo. Esta aba, pode ser acessada através do item na barra lateral, logo abaixo do mapa. A sua estrutura foi montada de maneira simples onde apenas são listados estes eventos que não puderam ser exibidos dentro do mapa da página principal.

Devido ao seu design simples, os problemas enfrentados nesta etapa estavam um pouco mais relacionados com questões mais globais do projeto. O maior problema esteve relacionado como a maneira em que a identificação de eventos diferentes era feito dentro do banco de dados. Isso porque a princípio, acreditava-se que o *link* de cada um dos *posts* seriam distintos entre si, o que possibilitaria uma identificação fácil a ser feita. Porém, por algumas vezes, os endereços dos *posts* se alteravam entre execuções do *scraping* de dados, o que acabava sendo interpretado como postagens diferentes e, consequentemente, eventos basicamente duplicados. Este ponto acabou sendo bem evidenciado dentro da página de eventos sem localização, como ilustrado na figura abaixo:

A maneira para contornar este tipo de problema foi alterar a maneira como estes eventos eram construídos, já que a identificação era feita a partir de um *hash* que se baseava no *link* para sua construção. Dessa forma, ao esbarrar no problema de mesmos eventos possuírem endereços distintos em alguns momentos, foi abordada uma estratégia para considerar, tanto todo o conteúdo do *post* quanto a sua rede social de origem, para a construção do *hash* identificador do evento.



Figura 5.11: Eventos duplicados

5.3 Controle de usuários

O sistema permite que novos usuários acessem e interajam com a plataforma de forma simples e direta sem qualquer necessidade de cadastro prévio, dessa forma, como citado na seção inicial deste capítulo, ao navegar pela aplicação, é possível buscar e filtrar diferentes eventos que estejam próximos de acontecer. Entretanto, o principal intuito do projeto era deixar o sistema personalizável e adaptável o máximo possível a partir da inserção de páginas de interesse do usuário, esta funcionalidade será descrita com mais detalhes na próxima seção.

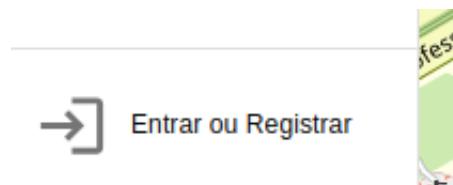


Figura 5.12: Botão para abrir janela de cadastro ou login

Por tal motivo, o processo de registro e *login* de usuários é essencial para que seja possível manter controle das preferências de cada usuário separadamente. A opção de se registrar ou entrar como um usuário se encontra no canto inferior esquerdo da tela. Ao clicar neste botão, uma nova janela com duas abas é aberta, sendo uma destinada para a entrada de usuários já cadastrados, e a outra, para cadastro de novos usuários no sistema.

Estes formulários contém campos necessários para a criação de um registro de usuário, como nome de usuário e senha por exemplo, esta última que é armazenada de forma criptografada no banco de dados. Após o preenchimento das informações, o sistema realiza uma chamada **HTTP** para o *back-end* para armazenamento das informações na plataforma.

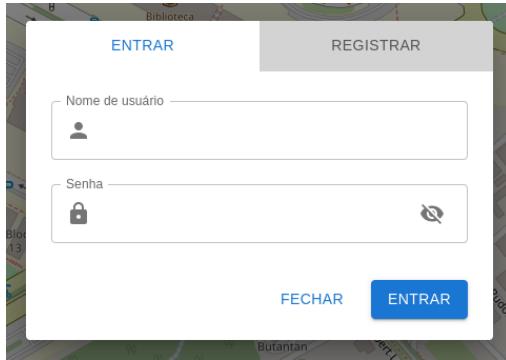


Figura 5.13: Janela de login de usuários

A imagem mostra uma interface de usuário para cadastro de novo usuário. No topo, há botões 'ENTRAR' e 'REGISTRAR'. Abaixo, campos para 'Nome de usuário', 'Email', 'Senha' e 'Nome' são exibidos. O campo de nome inclui um ícone para alternar visibilidade. No fundo, uma barra lateral mostra uma miniatura de mapa com rotas e endereços.

Figura 5.14: Janela de cadastro de novo usuário

5.4 Cadastro de páginas

Após o processo de registro e *login*, o sistema possibilita que os usuários cadastrem novas páginas que sejam de sua preferência e de onde gostariam que fossem obtidas as informações necessárias para construção dos eventos que serão mostrados na tela principal.

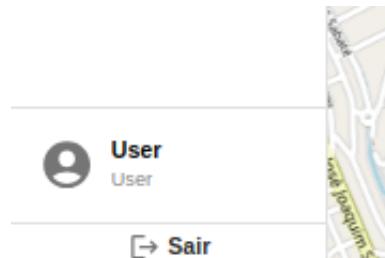


Figura 5.15: Botão de acesso à área do usuário

Esta funcionalidade pode ser acessada através do botão na barra lateral contendo o ícone de identificação e o nome do usuário logado, como mostra na figura acima. Com isso, o usuário é redirecionado à página de configurações.

Inicialmente esta página se encontra vazia, apenas com um esqueleto de tabela e um botão para cadastrar novas páginas na parte superior direita. Ao clicar neste botão, será aberto um formulário onde o usuário será capaz de registrar suas páginas de preferência a partir de onde deseja que seus eventos sejam obtidos.

Neste formulário, existe a possibilidade de cadastrar páginas tanto do *Facebook* quanto do *Instagram*. O sistema faz uma verificação através de expressões regulares para garantir que o texto inserido esteja no formato de uma página *web* e que pertença à rede social escolhida na seleção.



Figura 5.16: Formulário de registro de webpages

Configurações		
Preferências de usuário ⓘ		
Link da página	Origem	Ações
🔗 facebook.com/prg.usp	Facebook	
🔗 instagram.com/imewolves	Instagram	

Figura 5.17: Aparência da página com webpages cadastradas

A visualização das páginas é possível apenas para os usuários que fizeram o seu registro, ou seja, cada página cadastrada é única para cada usuário diferente, o que torna a experiência de cada pessoa personalizável aos seus próprios gostos e interesses. Vale ressaltar o fato de que apesar da unicidade das páginas para cada usuário, ainda é possível que dois usuários diferentes insiram uma mesma página nas suas preferências, por isso, a relação `WebPageUsers` criada no banco de dados é importante, já que é através dela que a associação de um usuário à uma `webpage` criada por ele pode ser identificada.

Ao cadastrar novas páginas, é iniciada a construção de uma lista na tabela que fica disponível para visualização exclusiva do usuário, ou seja, apenas o responsável pelo cadastro da página em questão consegue visualizar seu registro na tabela.

A partir do cadastro de suas páginas de preferência, os usuários são capazes de realizar a edição ou a remoção das informações que não tenham mais interesse através dos ícones de ação localizado em cada uma das linhas da tabela. Essas ações sobre cada uma das `webpages` são controladas por meio de APIs feitas no *back-end* do projeto, estas necessitam que um *token* de autorização seja passado nas requisições HTTP para diferenciar cada uma das páginas e cada um dos usuários que as cadastraram.

Apesar do desenvolvimento desta etapa ter sido desafiador, grande parte dos problemas enfrentados e as decisões tomadas para contorná-los já foram destacados em outras seções. Na seção `Amostragem de eventos`, por exemplo, é citado que um dos principais empecilhos



Figura 5.18: Ícones de ação de webpages

enfrentados no projeto foi a dificuldade em se obter os dados através do processamento em tempo de execução, e este problema é realçado dentro da funcionalidade descrita nos parágrafos anteriores, já que, mesmo o sistema permitindo que o registro e alteração das páginas possa ser feito à qualquer momento, essas mudanças serão perceptíveis, em termos da visualização dos eventos pelo usuário, em até 6 horas no pior caso, já que este foi o intervalo escolhido entre cada execução automatizada do processo de obter os eventos.

5.4.1 Controle de páginas para usuários diferentes

O maior empecilho enfrentado nesta seção, foi em relação ao controle de *webpages* iguais para diferentes usuários. Como descrito nesta seção, o sistema permite que uma mesma página seja cadastrada durante interações distintas de diferentes usuários. Entretanto, neste cenário, torna-se necessário que o controle para ambas as pontas sejam independentes entre si, ou seja, mudanças realizadas por um usuário, como uma edição ou uma deleção da *webpage* por exemplo, devem ser refletidas apenas para quem as fez. O problema relacionado à este ponto se deu pela maneira como as APIs estavam construídas, de forma que este comportamento ideal não estava sendo mantido. A princípio quando dois usuários possuíam um cadastro da mesma página e um deles realizava alguma ação sobre ela, a mudança era refletida na outra ponta indevidamente.

O problema ocorria devido ao modo que as APIs interagiam com os dados do banco para poder realizar as ações solicitadas pelo usuário, isso porque, mesmo com a existência da relação *WebPageUsers*, a ação era realizada sobre a própria *webpage* ao invés de agir apenas sobre o registro da ligação intermediária entre o usuário que realizou a ação e a página, dessa forma, do ponto de vista do usuário que teve seu registro alterado incorretamente, sua referência era mantida porém a entidade referenciada havia sido alterada o que causava a mudança indevida do seu registro.

O processo de correção deste problema levou à criação de algumas tarefas no quadro de atividades para que fossem priorizados, já que era um comportamento que afetaria a qualidade do sistema em produção e, apesar de a solução parecer simples, a implementação trouxe alguns empecilhos de implementação, principalmente causados pela falta de experiência com o *framework* utilizado no controle das APIs. Além disto, a correção para as ações de deleção e edição também diferiam entre si, sendo necessário uma abordagem de reparo distinto para cada método.

Capítulo 6

Pontos de melhoria

O sistema cumpre o que foi proposto, mas em desenvolvimento de *software*, sempre há margens para aprimoramentos. A seguir, serão listados alguns pontos de melhoria identificados ao longo da execução do projeto.

6.1 Cadastro de eventos pelos usuários

Uma das grandes etapas do funcionamento do sistema é a raspagem de dados de *websites* terceiros e isso traz consigo alguns problemas difíceis de serem evitados como:

- **Suspensão de uso:** caso a plataforma seja expandida para comportar *webscraping* em outros *sites*, é importante levar em conta que alguns têm termos de serviço que explicitamente proíbem essa prática. Isso pode levar ao bloqueio do acesso aos dados por completo e até ações legais contra o responsável.
- **Instabilidade na estrutura HTML:** esse processo é muitas vezes sensível a alterações na estrutura e *design* do *site*, exigindo atualizações frequentes para manter a funcionalidade.
- **Dependência de disponibilidade:** *webscraping* depende da disponibilidade contínua do *site* de origem. Caso ele esteja fora do ar ou passe por manutenção, a coleta de dados será interrompida.

Esses são alguns bons motivos para tornar o sistema construído independente dessa etapa. Para tanto, é possível adaptar o *Back-end* e o *Front-end* para comportar o cadastro de eventos pelos próprios usuários da plataforma.

Essa mudança traz vantagens imediatas, como dispensar a fase de processamento textual, melhorando a qualidade das informações disponibilizadas e removendo a dependência com o serviço *Hugginface chatbot*. Além disso, o evento já fica visível aos usuários logo em seguida de seu registro, diferentemente de como o sistema está construído, sendo necessário esperar 6 horas até a próxima execução da raspagem de dados.

6.2 Sistema em contêiner Docker

Na configuração atual, o *Front-end*, *Back-end* e os *scripts* de raspagem de dados são simplesmente executados em segundo plano no servidor. Embora isso atenda às necessidades do projeto acadêmico, a implementação poderia se beneficiar significativamente ao incorporar a tecnologia Docker para gerenciar esses componentes.

Docker é uma plataforma de código aberto que permite automatizar processos de *deploy*, escalabilidade e gerenciamento de diferentes aplicações no que é chamado de contêineres. Eles são ambientes capazes de executar aplicações e gerenciar suas dependências de maneira encapsulada, sem interferência direta com outros processos rodando dentro da máquina ou outros contêineres, permitindo que elas funcionem em qualquer ambiente.

A utilização do Docker traz algumas vantagens imediatas, como:

- **Fluxo de desenvolvimento:** caso outros desenvolvedores contribuam com o projeto, Docker permite que seja possível construir a aplicação em um ambiente próximo ao de produção. Isso previne problemas de compatibilidade entre máquinas e facilita a colaboração de diferentes desenvolvedores.
- **Isolamento:** o fato do Docker executar aplicações em contêineres separados, fornece isolamento entre todas as partes do sistema. Isso significa que problemas de dependência em pacotes compartilhados entre os *scripts* e o *back-end* são improváveis de acontecer.
- **Escalabilidade:** caso o *back-end* exija mais recursos em situações de grande demanda, por exemplo, é possível simplesmente escalar horizontalmente a aplicação executando mais um contêiner *back-end*.

Esses são alguns benefícios a longo prazo que a utilização do Docker pode trazer nessa plataforma.

Capítulo 7

Conclusão

Ao final deste projeto, foi desenvolvida uma plataforma capaz de compilar informações a respeito de eventos de caráter cultural e esportivo que vão ocorrer dentro do ambiente universitário da USP, além de permitir a possibilidade de personalização com base no interesse dos usuários para que sejam exibidas apenas informações relevantes e de preferência do mesmo. A plataforma pode ser acessada através do endereço: <https://uspevents.ix.tc/>.

Vale destacar que a partir do ideal proposto no início do projeto, as expectativas puderam ser cumpridas, tendo em vista os pontos citados no parágrafo anterior. Mesmo com todos os problemas e desafios que foram enfrentados e foram relatados, principalmente pelo capítulo anterior, mas também ao longo de todos os outros, onde pode se destacar todos os empecilhos relacionados com a raspagem de dados a partir de redes sociais, foi possível realizar execuções completas de todo o processo automático descrito na seção de fluxo da plataforma em algumas ocasiões, começando desde a busca de postagens feitas nas páginas cadastradas, passando pela etapa de extração das informações a partir do processamento textual, até a criação e exibição do evento dentro da plataforma.

É viável de se afirmar que o projeto como um todo teve de grande importância para se aplicar conhecimentos obtidos durante toda a graduação e que também são de grande utilidade e frequentemente utilizadas durante o dia a dia de um profissional na área de desenvolvimento de software, já que, ao longo do trabalho, assim como destacado durante este documento, as intenções de se manterem boas práticas de desenvolvimento foram mantidas tanto no caráter técnico, com código limpo, testes unitários e uma arquitetura simples e concisa, quanto no aspecto não técnico, envolvendo aplicações das metodologias ágeis e alinhamentos entre os membros sobre as prioridades e os pontos mais importantes a serem desenvolvidos.

Assim como relatado no capítulo anterior, existem pontos dentro do projeto que podem ser melhorados e que poderiam ser explorados durante um desenvolvimento futuro, de forma que o sistema pudesse adquirir um caráter bem mais genérico, abrangendo outros tipos de regiões, ou aperfeiçoando-o até um ponto que possa ser usado por outras instituições públicas como prefeituras, por exemplo. Além disso, novas abordagens poderiam ser exploradas para contornar a grande dificuldade de se trabalhar com raspagem de dados e toda a parte de dependências de sistemas terceiros à plataforma.

Referências

- [Agile Manifesto 2001] Agile Manifesto. <https://agilemanifesto.org>. Accessado em: 2023-11-08. 2001 (citado na pg. 7).
- [ASTIN *et al.* 1997] Alexander W ASTIN *et al.* *What matters in college?* JB, 1997 (citado na pg. 1).
- [BIGELOW 2023] Stephen BIGELOW. *10 prompt engineering tips and best practices*. Accessed: 11/23/2023. 2023. URL: <https://www.techtarget.com/searchenterpriseai/tip/Prompt-engineering-tips-and-best-practices> (citado na pg. 17).
- [MARTIN 2008] Robert C MARTIN. “Clean code: a handbook of agile software craftsmanship.(2008)”. *Citado na (2008)* 19 (2008) (citado na pg. 9).
- [TINTO 2012] Vincent TINTO. *Leaving college: Rethinking the causes and cures of student attrition*. University of Chicago press, 2012 (citado na pg. 1).
- [WAKODE *et al.* 2015] Rajat B WAKODE, Laukik P RAUT e Pravin TALMALE. “Overview on kanban methodology and its implementation”. *IJSRD-International Journal for Scientific Research & Development* 3.02 (2015), pp. 2321–0613 (citado na pg. 7).