

# Sistema Gerenciador de Eventos: Facilitando a Busca de Atividades para Usuários em Geral <sup>1</sup>

Gabriel Farinela<sup>2</sup>, Felipe Martin Sampaio<sup>3</sup>, Rafael Coelho<sup>4</sup>

Instituto Federal do Rio Grande do Sul - Campus Farroupilha  
Tecnologia em Análise e Desenvolvimento de Sistemas  
95174-274 – Farroupilha – RS – Brasil

gabrielfarinela@hotmail.com, felipe.sampaio@farroupilha.ifrs.edu.br,  
rafael.coelho@farroupilha.ifrs.edu.br

**Resumo.** *Este trabalho apresenta uma proposta de sistema gerenciador de eventos, cujo objetivo principal é facilitar a busca por eventos no geral, desde atividades para fins pessoais, até programas para fins profissionais. O sistema é uma plataforma web projetada para atender às necessidades de busca dos usuários que estão em busca de novas atividades. As tecnologias utilizadas no desenvolvimento foram os frameworks React e Next.js, junto ao MongoDB: o React foi empregado para a criação das interfaces, o Next.js para o desenvolvimento de uma API, e o MongoDB como banco de dados não relacional, utilizado para armazenar e disponibilizar informações sobre os usuários e os eventos do sistema. Como resultado do trabalho desenvolvido, foi definido e concluído um MVP do sistema, abrangendo as principais funcionalidades e permitindo o cumprimento dos objetivos previamente estabelecidos.*

**Abstract.** *This work presents a proposal for an event management system, whose main objective is to facilitate the search for events in general, ranging from personal activities to professional programs. The system is a web platform designed to meet the search needs of users looking for new activities. The technologies used in the development were the React and Next.js frameworks, along with MongoDB: React was employed for creating the interfaces, Next.js for developing an API, and MongoDB as a non-relational database used to store and provide information about users and system events. As a result of the work carried out, an MVP of the system was defined and completed, encompassing the main functionalities and meeting the previously established objectives.*

---

<sup>1</sup> Artigo científico referente ao Trabalho de Conclusão de Curso do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal do Rio Grande do Sul - *Campus* Farroupilha.

<sup>2</sup> Aluno matriculado no Trabalho de Conclusão de Curso.

<sup>3</sup> Professor co-orientador do Trabalho de Conclusão de Curso.

<sup>4</sup> Professor orientador do Trabalho de Conclusão de Curso.

## 1. Introdução

Hoje em dia, a disponibilidade de informações sobre eventos de interesse do usuário é muitas vezes disperso, estes eventos que vão de atividades pessoais, onde o indivíduo esteja buscando algum tipo de experiência ou até programas de lazer, dificulta a busca pontual e precisa que realmente atenda suas expectativas e preferências individuais. No contexto deste trabalho, foi identificada uma demanda da sociedade atual em buscar formas de encontrar atividades valiosas para seu bem estar mental e físico, a fim de resolver dificuldades na vida pessoal e profissional. Esta sobrecarga de informações em um mundo digital é um problema que afeta diversas áreas da vida. A busca por experiências e a constante conectividade da sociedade tem intensificado a sensação de FOMO (*Fear Of Missing Out* - medo de estar perdendo algo). Este fenômeno está ligado à pressão social para estar presente em eventos relevantes, juntamente com a dificuldade de encontrar eventos que se encaixem e preencham esse sentimento negativo (MOURA, 2021).

Recentemente empresas como a Sympla (SYMPLA, 2024) e a Eventin (EVENTIN, 2024), a primeira buscando eventos musicais, desde shows nacionais até festivais internacionais, já a segunda com eventos de cinema, buscando atrair público para o cinema nacional, trazendo mais visibilidade para este cenário têm investido em plataformas capazes de atender a uma ampla variedade de buscas na internet. No entanto, essas plataformas têm um foco mais generalista, permitindo que os usuários encontrem suas preferências de forma aleatória ou apenas após um uso aprofundado. Neste trabalho, desenvolvemos uma aplicação web com o intuito de auxiliar a resolução exatamente deste problema, oferecendo uma busca mais específica e personalizada. Além disso, incentivamos que os usuários do sistema favoritem e compartilhem seus principais eventos, promovendo interações e criando conexões significativas, através da conexão estabelecida pela atividade realizada durante o evento. As pessoas não buscam apenas um meio de entretenimento, elas desejam se divertir, se descobrir e, conseqüentemente, serem descobertas.

O presente trabalho busca amenizar os efeitos de FOMO ao oferecer uma plataforma intuitiva e personalizada para o uso facilitado dos usuários, identificando eventos interessantes e alinhados ao seu perfil, reduzindo assim a ansiedade e estresse associados à tomada de decisão.

Este trabalho desenvolveu uma aplicação web que, utilizando a API do Google Gemini (GEMINI, 2024), oferece uma solução prática para usuários que buscam eventos específicos com base em suas categorias de interesse. A plataforma permitirá que o usuário interaja com diversos eventos, com informações detalhadas, como nome, mês de ocorrência, *link*, descrição e fotos, facilitando a escolha de atividades que mais se encaixam em seu perfil. Para o desenvolvimento do trabalho, foram utilizadas as linguagens JavaScript (JAVASCRIPT, 2024) e TypeScript (TYPESCRIPT, 2024), sendo a primeira usada para o *back-end* juntamente com o *framework* Next.js (NEXTJS, 2024), e a segunda com o *framework* React.JS (REACT, 2024). Para o banco de dados,

foi escolhido o sistema de gerenciamento MongoDB (MONGODB, 2024), por ser projetado para escalabilidade horizontal, o que nos auxiliará posteriormente ao lidar com mais servidores e grandes volumes de dados.

O sistema proposto visa não apenas simplificar a busca por eventos, mas também reduzir a ansiedade e o estresse que muitos usuários enfrentam ao tentar encontrar atividades que se alinhem aos seus interesses. Ao oferecer recomendações personalizadas, esperamos aumentar a satisfação dos usuários e fortalecer a comunidade de pessoas interessadas em eventos relacionados, contribuindo para um ambiente mais conectado e engajado.

O projeto se insere no campo da interação humano-computador, buscando criar uma interface intuitiva que favoreça a usabilidade e a experiência do usuário. Além disso, conceitos da ciência da computação, como algoritmos de recomendação e estrutura de dados, são fundamentais para o funcionamento do sistema. O *design* de informação também desempenha um papel importante, uma vez que a organização e visualização de dados impactam diretamente a eficácia da plataforma.

Este artigo está organizado da seguinte maneira. Na Seção 2, são apresentados conceitos essenciais sobre sistemas de recomendação, experiência do usuário (UX), tecnologias web modernas e integração com APIs, que embasam a construção da plataforma. A Seção 3 descreve as etapas e processos do desenvolvimento do sistema web, falando acerca das ferramentas, técnicas e abordagens utilizadas. Na Seção 4, exploramos as etapas de criação da aplicação, desde as escolhas de design até as funcionalidades implementadas. Por fim, a Seção 5 discute os principais resultados obtidos e as contribuições do sistema para a simplificação e personalização na busca por eventos, assim como seu impacto positivo na experiência do usuário.

## **2. Referencial Teórico**

Esta seção abordará tópicos relevantes para o embasamento teórico deste artigo, os quais são relacionados com a temática de sistemas de recomendação.

Os métodos tradicionais utilizados no desenvolvimento de sistemas de recomendação, como a filtragem colaborativa e a filtragem baseada em conteúdo, têm sido amplamente usados. A filtragem colaborativa recomenda itens com base nas avaliações de usuários semelhantes, enquanto a filtragem baseada em conteúdo sugere itens com base nas características dos próprios itens, comparando-os com o histórico de preferências do usuário (MOTTA, 2011). No entanto, uma abordagem alternativa, mais explícita, tem ganhado destaque: a utilização de preferências definidas diretamente pelo usuário.

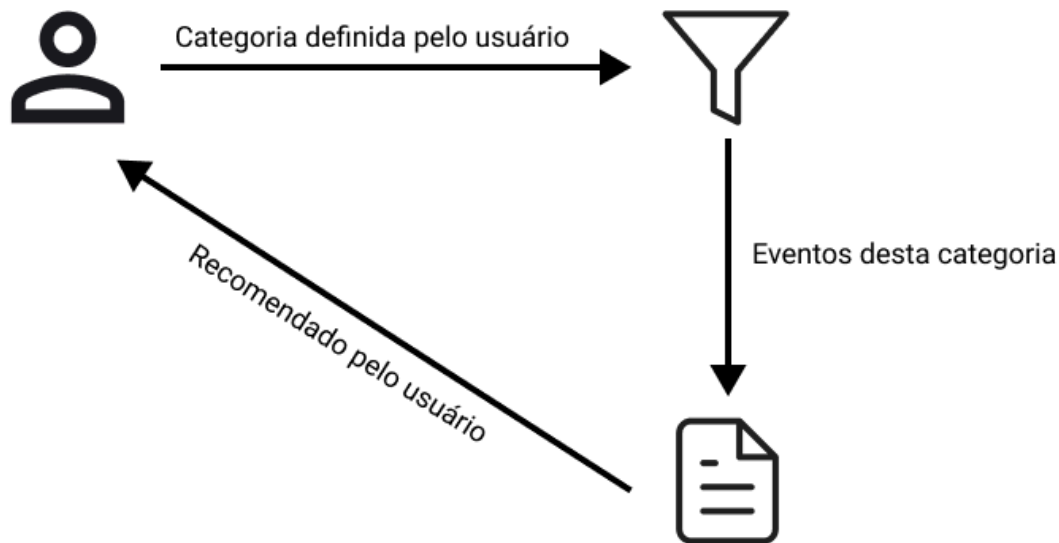
Essa abordagem se difere por permitir que o usuário selecione categorias de interesse de forma direta, ajustando o sistema para recomendar apenas itens dessas categorias. Isso reduz a complexidade do algoritmo, pois elimina a necessidade de

cálculos de similaridade entre usuários e itens, que podem demandar processamento intensivo. Além disso, essa opção oferece um modelo mais simples, com menores requisitos computacionais. Bobadilla et al. (2013) reforçam que sistemas baseados em preferências explícitas são particularmente eficientes em contextos onde o controle do usuário sobre a recomendação é prioritário. Eles também apontam que essa abordagem minimiza problemas como a superespecialização, frequentemente observada na filtragem baseada em conteúdo (BURKE, 2002).

Ao permitir que o usuário escolha diretamente as categorias de eventos que deseja, o sistema adota um modelo que prioriza as preferências explícitas, conforme discutido por Shani e Gunawardana (2011). Essa estratégia de personalização direta oferece uma experiência mais precisa, evitando as limitações de métodos que dependem de soluções baseadas em dados históricos ou comportamentais. Assim, o sistema se alinha à teoria de preferências explícitas, garantindo um alto grau de personalização.

Como ilustrado na Figura 1, o sistema desenvolvido neste trabalho apresenta uma aplicação prática de um modelo de recomendação que, ao focar nas preferências explicitamente declaradas pelo usuário, proporciona uma experiência mais eficiente e personalizada. O usuário, ao acessar seu perfil, define de forma direta sua preferência por categorias de eventos, sem depender de algoritmos de predição. Dessa maneira, o sistema recomenda apenas eventos relacionados às suas escolhas.

O problema do novo usuário em sistemas de recomendação, também conhecido como "cold start", ocorre quando há uma falta de informações iniciais sobre as preferências e o comportamento do usuário, dificultando a geração de recomendações personalizadas. Este desafio é particularmente significativo em contextos onde a experiência personalizada é crucial desde o primeiro contato do usuário com a plataforma. Para mitigar essa limitação, abordagens modernas têm sido propostas, como a utilização de métodos híbridos e inferências baseadas em dados implícitos ou características de personalidade. Conforme discutido por al Fararni et al. (2020), o uso de técnicas combinadas e a integração de múltiplos modelos podem melhorar significativamente a eficácia dos sistemas de recomendação em cenários de "novo usuário". Essas estratégias buscam construir perfis preliminares com base em comportamentos gerais ou informações contextuais, ampliando a capacidade do sistema de oferecer recomendações relevantes desde o início.



**Figura 1: Fluxo de recomendação de eventos de preferências explícitas**

**Fonte: Autoria própria.**

Shani e Gunawardana (2011) demonstram que a utilização de preferências explícitas é eficaz em situações em que os usuários possuem interesses bem definidos e desejam ter maior controle sobre o conteúdo recomendado. Tais sistemas oferecem benefícios significativos, especialmente em plataformas onde a precisão e a personalização são prioritárias, eliminando a necessidade de processar grandes volumes de dados históricos ou comportamentais, como ocorre na filtragem colaborativa.

Além disso, de forma a complementar, Schafer, Konstan e Riedl (2001) argumentam que sistemas baseados em recomendações diretas, definidas pelo próprio usuário, são mais eficientes e escaláveis. Como esses sistemas não dependem de complexos algoritmos de predição, eles proporcionam uma experiência fluida, com menor carga de dados e maior velocidade de resposta. Assim, a recomendação explícita se alinha à simplicidade e eficácia, refletindo diretamente as preferências do usuário, sem a necessidade de complexidade adicional no algoritmo.

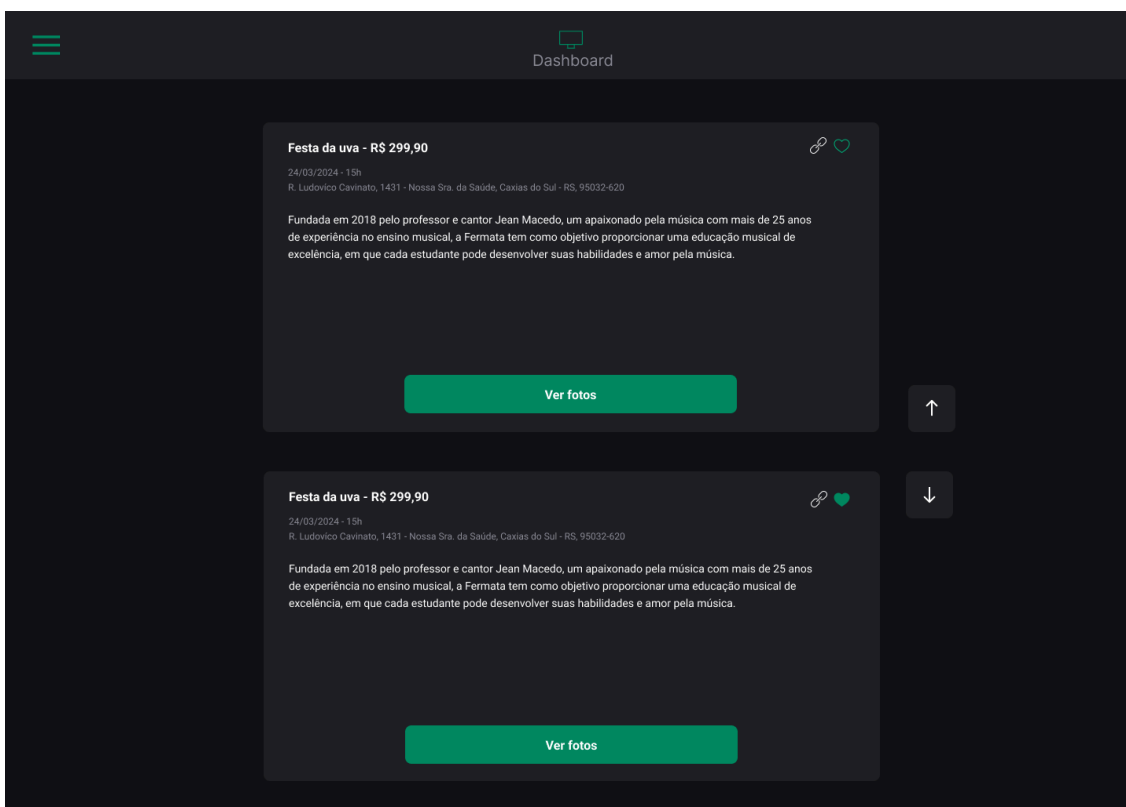
### **3. Metodologia**

Esta seção apresenta os tópicos relevantes e as metodologias adotadas no desenvolvimento deste projeto. Serão discutidas as experiências dos usuários com o sistema, destacando como elas estão diretamente relacionadas ao seu sucesso, além de destacar a importância da seleção adequada das tecnologias para sua implementação. Por fim, será justificada a escolha da API utilizada para a obtenção de informações sobre os eventos, explicando os motivos que fundamentam essa decisão.

### 3.1 Experiência do Usuário (UX)

A experiência do usuário final é determinante para a evolução e sucesso de qualquer aplicação. Don Norman, em *Designing for the Digital Age*, comenta sobre a importância da usabilidade e a atenção que tem que ser dada a *design* de interfaces intuitivas. Entre os pontos levantados estão o tempo de carregamento, facilidade de navegação e melhoria constante através de *feedbacks* dos usuários, através de postagens online, nas redes sociais do projeto.

Para este trabalho, a implementação de uma interface amigável e acessível é uma prioridade, com o objetivo de proporcionar uma experiência agradável ao usuário. Desta forma, aplicou-se uma ideia moderna, adotada primeiramente pelo TikTok (TIKTOK, 2024), um dos maiores aplicativos de vídeos desde sua criação, que utiliza um *feed* infinito. Este sempre mostra mais vídeos e, de forma simples, permite que o usuário continue assistindo com apenas a rolagem do dedo ou, na versão web, pressionando um botão. A Figura 2 apresenta este conceito aplicado à interface da aplicação desenvolvida neste trabalho.



**Figura 2: Layout do dashboard.**

**Fonte: Elaboração própria.**

Este *design* atual transmite a sensação de simplicidade, com poucas informações exibidas na tela atual, enquanto ainda incentiva o usuário a buscar o próximo conteúdo.

Esta abordagem tem como propósito engajar trazendo uma experiência fluida, e, ao mesmo tempo, reduzir os sintomas de FOMO.

### 3.2 Tecnologias adotadas

Para o desenvolvimento desta aplicação, foram adotadas algumas tecnologias modernas porém já muito bem estabelecidas e reconhecidas na área. Para o *front-end* foi utilizado o React, *framework* JavaScript desenvolvido pelo Facebook, atual Meta (META, 2024). Para uma maior segurança e estabilidade foi adicionado o TypeScript, que é uma linguagem baseada em JavaScript com alguns recursos de linguagem adicionais, como interfaces e tipos de dados. Com relação ao *back-end* da aplicação, será usado o Next.js, que é um *framework* JavaScript e de fácil integração com aplicativos React e APIs. Além disso, o uso do MongoDB para o armazenamento de dados permitirá consultas rápidas e eficientes, alinhando-se às necessidades de performance do sistema.

### 3.3 Integração com APIs

A integração com a API do Google Gemini é um ponto central no desenvolvimento do projeto. Através desta API, o sistema poderá acessar informações relevantes sobre eventos, como descrições e links. Compreender o funcionamento das APIs, incluindo métodos de consumo de dados e considerações sobre segurança, é vital para garantir uma integração bem-sucedida.

Para compreender como a API Gemini foi utilizada, é essencial entender seu funcionamento básico. Uma API atua como um intermediário, permitindo a comunicação entre diferentes sistemas. No atual caso, ela funciona como um canal que possibilita ao sistema acessar e utilizar informações de maneira eficiente. Essa API utiliza modelos de linguagem baseados em algoritmos complexos, treinados em grandes volumes de dados textuais, que permitem interpretar perguntas ou comandos e gerar respostas em linguagem natural, se aproximando da utilizada pelos humanos. A API oferece uma ampla gama de aplicações, desde a criação de chatbots até a geração de conteúdo criativo. No contexto deste projeto, sua principal funcionalidade utilizada foi o acesso à base de informações e a capacidade de realizar pesquisas em tempo real na internet. Além disso, os modelos são constantemente atualizados para garantir a maior precisão e relevância das informações fornecidas.

Como apresentado na Figura 3, inicialmente são importadas as classes e objetos da biblioteca `google/generative-ai`, incluindo a classe principal `GoogleGenerativeAI`, que é responsável por interagir com a API generativa, e dois objetos `HarmCategory` e `HarmBlockThreshold` usados para configurar filtros que evitam respostas impróprias. Em seguida, na linha 3, é armazenada a chave da API fornecida pelo Google, a qual é utilizada para autorizar e autenticar o serviço. Com essas informações armazenadas em

propriedades, uma instância da classe `GoogleGenerativeAI` é criada utilizando a chave da API. Esta instância será usada posteriormente para acessar e configurar os modelos.

```
1 import { GoogleGenerativeAI, HarmCategory, HarmBlockThreshold } from "@google/generative-ai";
2
3 const apiKey = "AIzaSyC--W0huoP1UGaSL2xFyqAw0SfWTGyrTsk";
4 const genAI = new GoogleGenerativeAI(apiKey);
5
6 const model = genAI.getGenerativeModel({
7   model: "gemini-1.5-flash",
8 });
9
10 const generationConfig = {
11   temperature: 0,
12   topP: 0.95,
13   topK: 64,
14   maxOutputTokens: 20000,
15   responseMimeType: "application/json",
16 };
17
18 const safetySettings = [
19   { category: HarmCategory.HARM_CATEGORY_HARASSMENT, threshold: HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE, },
20   { category: HarmCategory.HARM_CATEGORY_HATE_SPEECH, threshold: HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE, },
21   { category: HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT, threshold: HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE, },
22   { category: HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT, threshold: HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE, },
23 ];
```

**Figura 3: Código de configuração do modelo generativo.**

**Fonte: Elaboração própria.**

Com a classe já instanciada, o próximo passo é configurar o modelo generativo específico a ser utilizado. No presente projeto, o modelo escolhido foi o `gemini-1.5-flash`, que é especializado em fornecer respostas rápidas e contextualizadas, além de ser capaz de lidar com tarefas de alta demanda e processamento rápido.

A seguir, são definidas as configurações que controlam o comportamento e o formato das respostas geradas pelo modelo. Essas configurações incluem:

- `temperature`: controla a aleatoriedade das respostas geradas pelo modelo. Valores próximos de zero tornam as respostas mais determinísticas e precisas, o que é ideal para tarefas previsíveis.
- `topP`: limita a probabilidade cumulativa de seleção das próximas palavras e contextos. Por exemplo, um valor de 0.95 significa que apenas as palavras com 95% da probabilidade total serão consideradas.
- `topK`: limita a seleção das palavras mais prováveis em cada etapa de geração. Valores menores resultam em respostas mais previsíveis e menos criativas.
- `maxOutputTokens`: define o número de *tokens* que a resposta gerada terá. Neste caso, 20.000 *tokens* é um valor elevado, útil para respostas longas, como nos casos de eventos.
- `responseMimeType`: define o formato da resposta gerada.

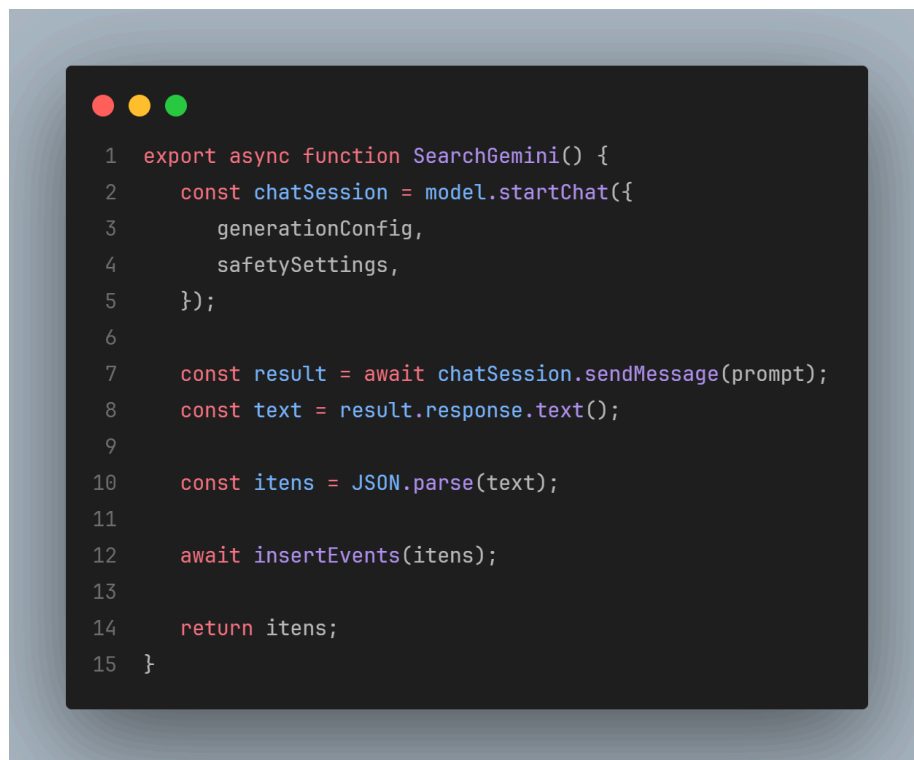


A partir da linha 18, o código configura as opções de segurança da API, com o objetivo de bloquear ou filtrar conteúdos nocivos. Para isso, a variável *HarmCategory* define as categorias de conteúdo prejudicial que o modelo deve monitorar e bloquear. As categorias definidas são:

- `harm_category_harassment`: conteúdos de assédio, como agressões verbais, intimidações ou abusos online.
- `harm_category_hate_speech`: discursos de ódio, que incluem mensagens que incitam violência ou discriminação contra grupos específicos.
- `harm_category_sexually_explicit`: conteúdos sexualmente explícitos ou inapropriados.
- `harm_category_dangerous_content`: conteúdos perigosos, como informações que incentivam atividades ilegais ou autolesivas.

Essas configurações são uma forma de garantir que o sistema não gere ou permita a propagação de conteúdos sensíveis ou perigosos, proporcionando uma boa experiência ao usuário e evitando a censura excessiva, conforme as diretrizes do próprio Google.

Já como apresentado na Figura 4, é criada uma sessão de *chat* com o modelo Gemini, utilizando as configurações anteriormente definidas, iniciando assim a comunicação com o modelo para realizar a busca. Após o início da sessão, o código envia uma mensagem solicitando ajuda com base no comando atual. Em seguida retorna o texto que é convertido de volta para um formato de dados mais utilizado, o JSON.



```
1  export async function SearchGemini() {
2      const chatSession = model.startChat({
3          generationConfig,
4          safetySettings,
5      });
6
7      const result = await chatSession.sendMessage(prompt);
8      const text = result.response.text();
9
10     const itens = JSON.parse(text);
11
12     await insertEvents(itens);
13
14     return itens;
15 }
```

**Figura 4: Função para buscar eventos utilizando modelo generativo Gemini.**

**Fonte: Elaboração própria.**

## **4. Desenvolvimento do Sistema**

Nesta seção, serão apresentados os resultados obtidos ao longo do desenvolvimento do projeto. O processo iniciou-se com a identificação dos requisitos necessários, que serviram como base para o estabelecimento dos objetivos principais do sistema. Após essa etapa, deu-se início ao desenvolvimento, estruturado em duas grandes áreas: o *back-end*, responsável pela lógica e funcionalidades internas do sistema, e o *front-end*, que engloba a construção da interface e interação com o usuário. Por fim, serão exibidos os resultados finais das interfaces desenvolvidas, destacando suas características específicas e suas respectivas funcionalidades.

### **4.1 Requisitos da Aplicação**

A base da aplicação está relacionada à criação de um usuário. Desta forma, cada utilizador do sistema terá que primeiramente criar uma conta e, então, realizar o *login* no sistema. Após isso, ele terá liberdade para redefinir sua senha no próprio perfil, além de definir a categoria de eventos que deseja ver no *dashboard*. Na área referente à visualização dos eventos, haverá a opção de favoritar um evento específico. Como resultado desta interação, em uma nova tela, o usuário poderá verificar a listagem de seus eventos salvos como favoritos.

Cada evento é uma entidade única, com suas informações próprias e pré-definidas. O usuário poderá, além de verificar suas informações no próprio *card*, ver fotos do evento pesquisadas em tempo real na web através da API do Google Imagens. O sistema tem como foco o encontro de eventos com base em categorias definidas de forma explícita no perfil do usuário. Além disso, também é possível rever posteriormente os eventos que mais o agradaram e conseguir acessar informações mais completas sobre um determinado evento.

### **4.2 Desenvolvimento do *back-end***

Primeiramente, definiu-se qual banco de dados seria utilizado, sendo escolhido o MongoDB Atlas, um sistema de gerenciamento de banco de dados multi-nuvem que oferece uma plataforma de dados flexível. Ele permite armazenar dados heterogêneos em documentos BSON. Com essa escolha, foi estruturado o banco de dados: criou-se um agrupamento (*cluster*), e definiram-se algumas coleções (*collections*). Foram necessárias três coleções: uma para os usuários, outra para os eventos e uma terceira para relacionar essas duas entidades.

O *back-end* foi desenvolvido em Next.js. Inicialmente, realizou-se a conexão com o banco de dados Atlas por meio de uma URL de ligação, utilizando o Mongoose (MONGOOSE, 2024), uma biblioteca de Modelagem de Dados de Objeto (ODM), que

facilita a modelagem dos dados. Após a implementação e os testes confirmando a conexão entre a aplicação e o MongoDB Atlas, os esquemas (*Schemas*) das coleções foram desenvolvidos. Em seguida, criaram-se os controladores (*controllers*), responsáveis por processar as requisições do *front-end*. Com os pontos de acesso a API (*endpoints*) criados e configurados para receber essas requisições, o *back-end* foi concluído.

No desenvolvimento do sistema, foi utilizada a API Gemini, para realizar a busca de eventos na web com base em prompts pré-definidos por categorias. Atualmente, são buscados eventos de dezenove categorias diferentes, que são então armazenados diretamente no banco de dados. Essa abordagem foi adotada devido à lentidão observada quando o sistema realizava buscas de eventos em tempo de execução, à medida que os usuários navegam pela aplicação, o que compromete o desempenho e a eficiência operacional. Com a busca antecipada e o armazenamento dos eventos, o sistema torna-se mais ágil e eficiente. Além disso, é possível implementar uma rotina periódica para remover eventos antigos e buscar novos a cada mês, garantindo que as informações permaneçam atualizadas.

Para o carregamento das imagens dos eventos, foi integrada a API do Google Imagens, criando um mecanismo de pesquisa programável (GOOGLESEARCH, 2024), utilizando uma pesquisa customizada baseada no título de cada evento. Essa busca retorna imagens de forma dinâmica, e, ao contrário dos eventos, as imagens não são armazenadas no banco de dados. Sempre que o usuário acessa as imagens de um evento, uma nova busca é realizada, permitindo a exibição de conteúdos atualizados e variados a cada consulta.

### **4.3 Desenvolvimento do *front-end***

O *front-end* do sistema foi desenvolvido utilizando React.js, com a integração do Vite como ferramenta de *build*. O Vite foi escolhido por ser uma tecnologia atual, amplamente utilizada no mercado e famosa por sua agilidade. Ele utiliza o ESBuild para compilar e processar o código, garantindo inicializações rápidas e um eficiente sistema de *Hot Module Replacement* (HMR), o que melhora significativamente a experiência durante o desenvolvimento. Essa escolha permitiu maior produtividade e um fluxo de trabalho otimizado.

Durante o desenvolvimento, optou-se por não utilizar bibliotecas de componentes prontas, como Material-UI ou Ant Design. Todos os componentes foram criados manualmente, utilizando a biblioteca Styled-Components para a estilização. Dessa forma tínhamos maior controle sobre o *design* e a funcionalidade dos elementos, além de garantir a identidade visual exclusiva do sistema.

O desenvolvimento dos componentes foi realizado em etapas. Inicialmente, foram criados elementos reutilizáveis, como campos de texto, campos de opções e

botões. Esses componentes serviram como base para as demais telas, garantindo o reaproveitamento de código caracterizado pelo React.

Para o gerenciamento de estado, foram utilizados *hooks* nativos do React, como `useState` e `useContext`. Um dos destaques foi a implementação de um contexto global para notificações, responsável por informar o usuário sobre o sucesso ou erro das ações realizadas no sistema.

Além disso, foram criados componentes específicos para *feedback* visual em processos de carregamento. Esses componentes tinham como função principal bloquear ações dos usuários enquanto estavam sendo requisitados dados do *back-end*, como *login* ou a busca de eventos. Isso foi fundamental para melhorar a experiência do usuário e evitar interações desnecessárias durante o processamento.

Para armazenar informações não sensíveis do usuário, foram utilizados *cookies*<sup>5</sup> por meio da biblioteca Js-Cookie. Essa solução permitiu um acesso rápido a dados do usuário, contribuindo para a fluidez da aplicação. A escolha pela biblioteca Js-Cookie foi feita devido à sua simplicidade de uso e suporte abrangente em navegadores modernos.

O gerenciamento das rotas do sistema foi realizado com o `CreateBrowserRouter`, que proporcionou controle total sobre as rotas existentes e a proteção de dados. As rotas protegidas verificavam se o usuário estava autenticado antes de permitir o acesso, redirecionando para a página de entrar em caso de não autenticação. Também foi configurada uma página de erro para rotas inválidas, orientando o usuário a retornar para páginas existentes.

As telas do sistema foram desenvolvidas de maneira modular, iniciando pelas interfaces de *login* e cadastro. Na tela de cadastro, foram implementadas validações robustas que verificam se o e-mail inserido já está registrado no sistema. Caso seja detectado um e-mail duplicado, uma mensagem de erro é exibida, utilizando o contexto previamente configurado para garantir a clareza e a consistência na comunicação com o usuário. Após a conclusão bem-sucedida do cadastro, o usuário é automaticamente redirecionado para a tela de *login*, onde pode realizar sua autenticação, com seus dados recém-criados, para acessar o sistema de forma segura. Essa abordagem modular e validada melhora a experiência do usuário, minimizando erros e garantindo um fluxo de navegação contínuo e eficiente.

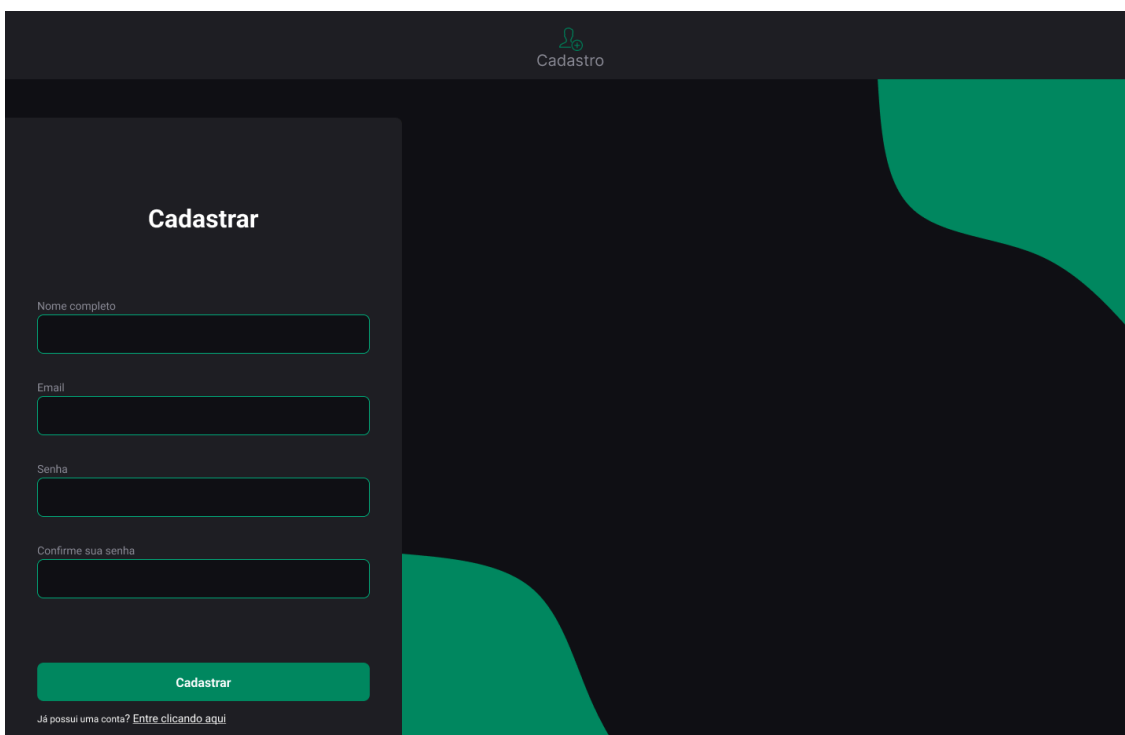
#### **4.4 Interfaces do Sistema**

Após a definição da metodologia e dos requisitos do sistema web, foram desenvolvidas as interfaces da aplicação. Os protótipos desenvolvidos foram pensados

---

<sup>5</sup> Arquivos de texto armazenados no navegador para lembrar preferências ou informações relevantes.

utilizando-se de *layout* atuais, como o do TikTok. Primeiramente foi desenvolvida a tela de cadastro de usuário, conforme apresentado na Figura 5.



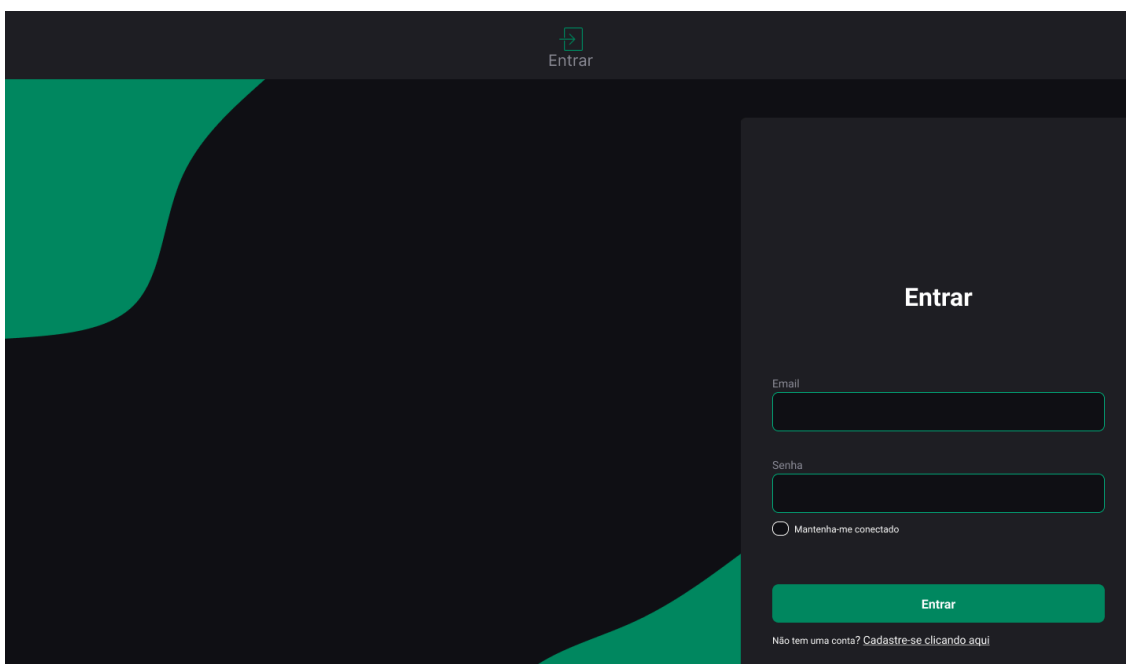
**Figura 5: Layout da tela de cadastro de usuário.**

**Fonte: Elaboração própria.**

Nesta interface foi adicionado um quadro no canto inferior esquerdo contendo os campos para o usuário preencher com suas informações pessoais básicas para acessar o sistema. Neste quadro de cadastro de usuário, há os campos: Nome Completo, E-mail, Senha e Confirme sua senha. Ao lado do quadro, há uma área livre, com duas imagens, cada uma posicionada em um canto da tela.

A Figura 6 apresenta a interface de *login* do sistema, desenvolvida com um *layout* similar à tela anterior. As únicas mudanças são que o quadro onde os campos de texto são armazenados está no canto inferior direito. Neste quadro, encontram-se os campos de e-mail e senha do usuário, novamente trazendo as duas imagens fixadas em cantos opostos da tela, na área livre de componentes.

Nas telas de cadastro e *login* de usuário, os campos de entrada de texto são validados para garantir que as informações fornecidas sejam corretas. Caso o usuário cometa um erro ao preencher os campos, a borda do campo, inicialmente configurada com a cor verde (indicando que a entrada está válida), será alterada para a cor vermelha. Isso visa alertar o usuário de que há um erro na informação inserida, ajudando-o a identificar e corrigir rapidamente o campo problemático.



**Figura 6: Layout da tela de entrar no sistema.**  
**Fonte: Elaboração própria.**

A Figura 7 apresenta a interface que tem por objetivo armazenar os eventos já favoritos pelo usuário, como uma lista em forma de *grid*, um evento após o outro será listado. Contendo o *card* do evento com as informações: título, data da atividade, local em que irá acontecer, categoria do evento e um botão para carregar as fotos do evento. Bem como o nome do evento, *link* para pesquisa, opção para remover dos favoritos, local e mês que o evento vai ocorrer, se tiver dados sobre valores, aparecerá ao lado do título. Além de um botão para abrir as imagens buscadas em tempo real sobre este evento.

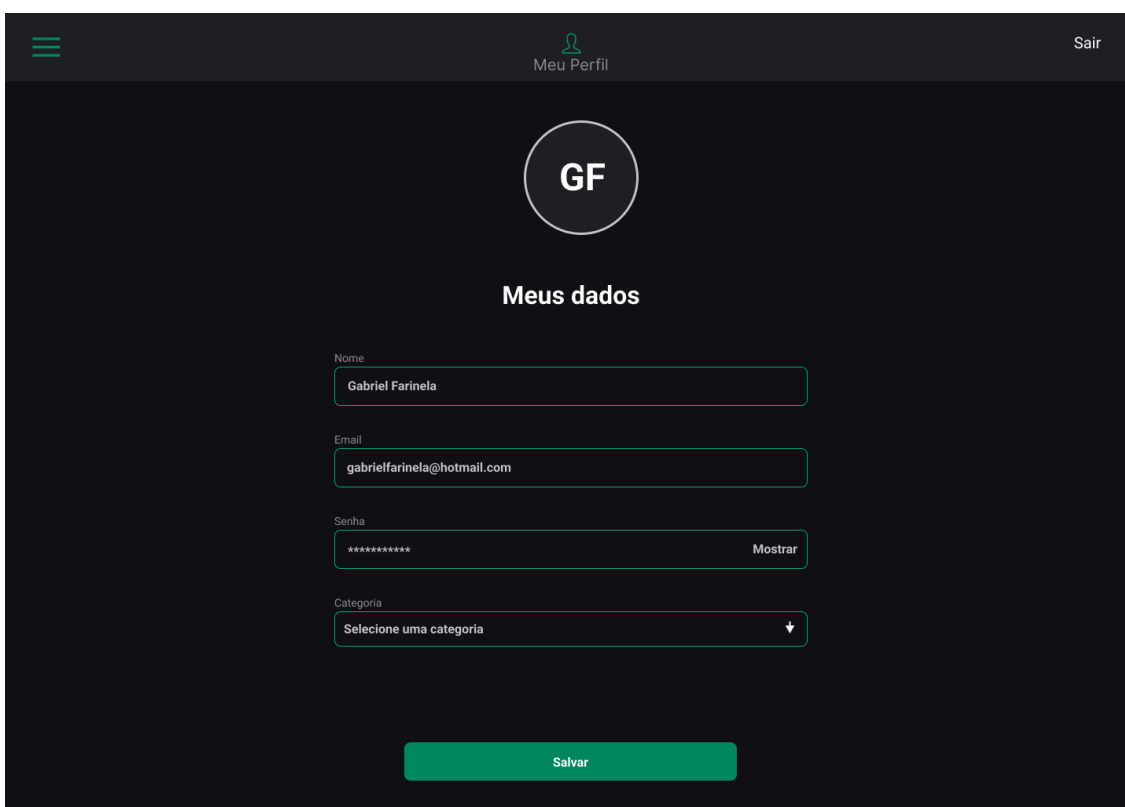


**Figura 7: Layout da tela de favoritos.**  
**Fonte: Elaboração própria.**

A tela de *dashboard*, como demonstrado na Figura 2, foi desenvolvida com base no *layout* proposto na prototipação, exibindo dois quadros por vez, cada um representando um evento com informações como nome, mês, local, categoria, *tags* e

*link*. Ao lado dos eventos, dois botões eram responsáveis por carregar mais eventos em tela, nem sempre sendo necessário requisitar no servidor, pois estas requisições eram chamadas somente se todos os eventos já carregados tiverem sido vistos.

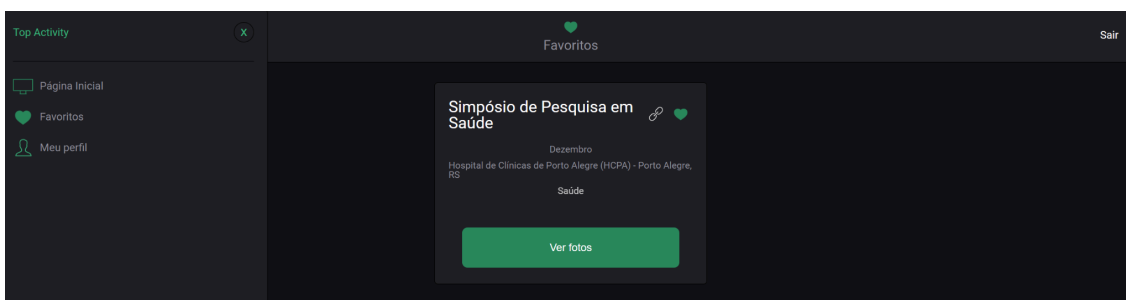
Na Figura 8, está representado o *layout* do perfil do usuário. Nesta tela, será possível atualizar o nome e a senha, sendo essa ação bloqueada para o campo de e-mail, que será único e utilizado também para auxiliar na validação de contas no *back-end*. O *layout* apresenta quatro campos definidos: Nome, E-mail, Senha e Categoria. O campo de Categoria será um campo de opções, cuja lista é buscada no banco de dados, permitindo ao usuário escolher a categoria pela qual deseja filtrar seus eventos.

The image shows a mobile application interface for a user profile. At the top, there is a dark header with a hamburger menu icon on the left, a user profile icon and the text 'Meu Perfil' in the center, and a 'Sair' button on the right. Below the header, a large white circle contains the initials 'GF'. Underneath this is the title 'Meus dados'. The form consists of four input fields: 'Nome' with the value 'Gabriel Farinela', 'Email' with the value 'gabrielfarinela@hotmail.com', 'Senha' with masked characters '\*\*\*\*\*' and a 'Mostrar' button, and 'Categoria' with a dropdown menu showing 'Selecione uma categoria'. At the bottom of the form is a large green button labeled 'Salvar'.

**Figura 8:** *Layout* da tela de perfil do usuário.

**Fonte:** Elaboração própria.

Por fim, definimos um menu, representado na Figura 9, para auxiliar o usuário na navegação entre as telas. Este menu apresenta as opções Página Inicial, Favoritos e Meu Perfil, cada uma direcionando para sua respectiva rota e carregando o componente correspondente.



**Figura 9: Layout do menu de páginas.**

**Fonte: Elaboração própria.**

## 5. Conclusões

Para finalizar este trabalho, podemos analisar os resultados alcançados, que reforçam a viabilidade do projeto e seu potencial de auxiliar pessoas em busca de eventos em geral. Esta aplicação web não se resume a uma simples listagem de atividades, seu objetivo principal é oferecer uma experiência personalizada e consistente, fundamentada em escolhas explícitas e individuais do próprio usuário.

Através da abordagem de recomendação adotada, nos afastamos das práticas tradicionais que utilizam algoritmos baseados em buscas turvas. Neste trabalho, buscamos aproximar o usuário, tornando-o o principal destaque ao definir suas preferências e a categoria que deseja explorar. Dessa forma, aumentamos a transparência, limitando os interesses individuais e reduzindo a sobrecarga cognitiva frequentemente associada ao FOMO. Ao devolver o controle ao usuário e oferecer uma plataforma transparente e acessível, este sistema propõe uma nova forma de lidar com as informações de cada usuário.

Assim, com este trabalho, esperamos incentivar futuras melhorias no mercado de eventos e de recomendação de buscas como mais abertura dos sites com quais informações eles estão pegando do usuário, além de tentar diminuir o número de dados que aparecem em tela, com o intuito de manter a atenção do usuário no sistema, além de promover novas discussões sobre o equilíbrio no uso dos dados dos usuários. Dessa forma, esperamos que o sucesso desta aplicação seja alcançado por meio do impacto positivo na comunidade e pelo futuro promissor que poderá ser gerado para aqueles que a utilizarem.

Entende-se que após a finalização deste trabalho o sistema poderá evoluir com novas funcionalidades, fomentando o *network* dos usuários e buscando tornar-se competitivo no mercado. Com o aumento do número de clientes, será necessário revisar os planos do atual banco de dados escolhido, pois o atual *Cluster* utilizado é a versão *free*, que tem algumas limitações como memória e cpu compartilhada, além de um servidor mais robusto.

Posteriormente poderá ser trabalhado com mais opções para o sistema de recomendação, além da forma explícita, poderemos trabalhar com a forma implícita.



Estudar casos de sucesso onde a informação concedida pelo usuário seja trabalhada de forma saudável, não assumindo riscos de privacidade e confiança.

Por fim, implementando um sistema de conversas entre dois usuários, poderia incentivar outros usuários a trazer seus quadros de eventos para o nosso sistema, além de investir em integração com outras plataformas de buscas.

## Referências

- BOBADILLA, J. et al. **Recommender systems survey**. *Knowledge-Based Systems*, v. 46, p. 109–132, 2013. DOI: <<https://doi.org/10.1016/j.knosys.2013.03.012>>. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0950705113001044>>. Acesso em: 29 nov. 2024.
- BURKE, R. **Hybrid Recommender Systems: Survey and Experiments**. *User Model User-Adap Inter*, v. 12, p. 331–370, 2002. Disponível em: <<https://doi.org/10.1023/A:1021240730564>>. Acesso em: 29 nov. 2024.
- CAZELLA, Sílvio César. **Sistemas de recomendação**. 2021. 21 p. Disponível em: <<https://abrir.link/uwusD>>. Acesso em: 29 nov. 2024.
- FARARNI, Khalid al; AGHOUTANE, Badraddine; RIFFI, Jamal; ABDELOUAHED, Sabri; YAHYAOU, Ali. **Comparative Study on Approaches of Recommendation Systems**. In: *Proceedings of the International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD)*. Singapore: Springer, 2020. p. 753-764. Disponível em: [https://doi.org/10.1007/978-981-15-0947-6\\_72](https://doi.org/10.1007/978-981-15-0947-6_72). Acesso em: 12 dez. 2024.
- GEMINI, **Começar a usar a API Gemini**, Google, 2024. Disponível em: <<https://ai.google.dev/gemini-api/docs?hl=pt-br>>. Acesso em: 29 nov. 2024.
- EVENTIN, Eventin, 2024. Disponível em: <<https://www.eventim.com.br/>>. Acesso em: 29 nov. 2024.
- SYMPLA, Sympla,, 2024. Disponível em: <<https://www.sympla.com.br/eventos/farroupilha-rs>>. Acesso em: 29 nov. 2024.
- GOOGLESEARCH, **Visão geral**, Google, 2024. Disponível em: <<https://developers.google.com/custom-search/docs/overview?hl=pt-br>>. Acesso em: 29 nov. 2024.
- JAVASCRIPT, **JavaScript**, MDN, Mozilla, 2024. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>>. Acesso em: 29 nov. 2024.

- META, Meta, 2024. Disponível em: <<https://www.meta.com/>>. Acesso em: 29 nov. 2024.
- MONGODB, **Documentação do MongoDB**, MongoDB Inc, 2024. Disponível em: <<https://www.mongodb.com/pt-br/docs/>>. Acesso em: 29 nov. 2024.
- MONGOOSE, **Getting Started**, MIT, 2024. Disponível em: <<https://mongoosejs.com/>>. Acesso em: 29 nov. 2024.
- MOTTA, C. L. R. et al. **Sistemas de recomendação**. Pimentel, M.; Fuks, H. “Sistemas colaborativos”. Rio de Janeiro: Elsevier, 2011.
- MOURA, Débora Ferreira et al. **Fear of missing out (FoMO), mídias sociais e ansiedade: Uma revisão sistemática**. *Psicologia, Conocimiento y Sociedad*, v. 11, n. 3, p. 99-114, 2021.
- NORMAN, Don. *The Design of Everyday Things*. Revised and expanded edition. New York: Basic Books, 2013.
- NEXTJS, **Introduction: Welcome to the Next.js documentation**, Vercel Inc, 2024. Disponível em: <<https://nextjs.org/docs>>. Acesso em: 29 nov. 2024.
- RASHID, A. M.; ALBERT, I.; COSLEY, D.; LAM, S. K.; MCNEE, S. M.; KONSTAN, J. A.; RIEDL, J. **Getting to Know You: Learning New User Preferences in Recommender Systems**. 2002. DOI: <<https://doi.org/10.1145/502716.502737>>. Acesso em: 29 nov. 2024.
- REACT, **React: The library for web and native user interfaces**, Facebook, 2024. Disponível em: <<https://react.dev/>>. Acesso em: 29 nov. 2024.
- SCHAFFER, J.B.; KONSTAN, J.A.; RIEDL, J. **E-Commerce Recommendation Applications**. *Data Mining and Knowledge Discovery*, v. 5, p. 115–153, 2001. DOI: <<https://doi.org/10.1023/A:1009804230409>>. Acesso em: 29 nov. 2024.
- SHANI, G.; GUNAWARDANA, A. **Evaluating Recommendation Systems**. In: RICCI, F.; ROKACH, L.; SHAPIRA, B.; KANTOR, P. (eds). *Recommender Systems Handbook*. Boston, MA: Springer, 2011. p. 257-297. DOI: <[https://doi.org/10.1007/978-0-387-85820-3\\_8](https://doi.org/10.1007/978-0-387-85820-3_8)>. Acesso em: 29 nov. 2024.
- TIKTOK, TikTok, TikTok, 2024. Disponível em: <<https://www.tiktok.com/>>. Acesso em: 29 nov. 2024.
- TYPESCRIPT, **TypeScript Documentation**, Microsoft, 2024. Disponível em: <<https://www.typescriptlang.org/pt/docs/>>. Acesso em: 29 nov. 2024.