UNIVERSIDADE FEDERAL DO PARÁ
CAMPUS UNIVERSITÁRIO DO TOCANTINS/CAMETÁ
FACULDADE DE SISTEMAS DE INFORMAÇÃO

JOSÉ WENDEL VIANA PANTOJA

APLICAÇÃO DE TÉCNICAS FRONT-END NO DESENVOLVIMENTO DE UMA PLATAFORMA WEB PARA DIVULGAÇÃO DE EVENTOS

CAMETÁ-PA
2025

JOSÉ WENDEL VIANA PANTOJA

APLICAÇÃO DE TÉCNICAS FRONT-END NO DESENVOLVIMENTO DE UMA PLATAFORMA WEB PARA DIVULGAÇÃO DE EVENTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação, Faculdade de Sistemas de Informação, Campus Universitário do Tocantins/Cametá, Universidade Federal do Pará, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Carlos dos Santos Portela.

CAMETÁ-PA
2025

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

V614a Viana Pantoja, José Wendel.
Aplicação de técnicas front-end no desenvolvimento de uma plataforma web para divulgação de eventos / José Wendel Viana Pantoja. — 2025.
xxx, 30 f. : il. color.

Orientador(a): Prof. Dr. Carlos dos Santos Portela
Trabalho de Conclusão (Graduação) - Universidade Federal do Pará, Campus Universitário de Cametá, Curso de Sistemas de Informação, Cametá, 2025.

1. Desenvolvimento web. 2. Front-end. 3. React. 4. Divulgação de eventos. I. Título.

CDD 005.11

JOSÉ WENDEL VIANA PANTOJA

APLICAÇÃO DE TÉCNICAS FRONT-END NO DESENVOLVIMENTO DE UMA PLATAFORMA WEB PARA DIVULGAÇÃO DE EVENTOS

Data da Defesa: Cametá-PA, 26 de Março de 2025.

Membros da Banca Examinadora:

gov.br
Documento assinado digitalmente
CARLOS DOS SANTOS PORTELA
Data: 28/04/2025 17:57:44-0300
Verifique em https://validar.iti.gov.br

Professor e orientador Carlos dos Santos Portela, Dr.

(Presidente) Universidade Federal do Pará

gov.br
Documento assinado digitalmente
ELTON SARMANHO SIQUEIRA
Data: 28/04/2025 10:51:02-0300
Verifique em https://validar.iti.gov.br

Prof. Elton Sarmanho Siqueira, Dr. (Membro interno)

Universidade Federal do Pará

gov.br
Documento assinado digitalmente
CESAR ELIAS DE CRISTO LOBATO
Data: 29/04/2025 00:26:29-0300
Verifique em https://validar.iti.gov.br

Cesar Elias de Cristo Lobato, Bel. (Membro externo)

Programa de Pós-Graduação em Computação Aplicada (PPCA)/UFPA

CAMETÁ-PA
2025

RESUMO

O avanço da tecnologia e da internet tem proporcionado mudanças significativas na maneira como os eventos são organizados e divulgados. Este trabalho, resultado de um estágio supervisionado, apresenta o desenvolvimento da plataforma *CommunityEvents*, um sistema web voltado para a centralização e divulgação de eventos, utilizando técnicas de desenvolvimento *Front-End*. A plataforma foi desenvolvida com React, Context API, Styled-Components e Ant Design, permitindo uma interface responsiva e eficiente. Além disso, utilizou-se o Firebase para gerenciar autenticação e armazenamento de dados, otimizando o tempo de desenvolvimento. A metodologia adotada envolveu a elicitação de requisitos, prototipação e implementação do sistema, seguindo princípios de boas práticas no desenvolvimento *Front-End*. Os resultados obtidos demonstraram a eficácia da plataforma na organização e apresentação de eventos, melhorando a experiência dos usuários na busca por informações. Como trabalhos futuros, propõe-se a implementação de um painel administrativo para maior controle sobre os eventos cadastrados.

Palavras-chave: Desenvolvimento Web. Front-End. React. Divulgação de Eventos.

ABSTRACT

The advancement of technology and the internet has significantly changed the way events are organized and promoted. This study, result of a supervised internship, presents the development of the *CommunityEvents* platform, a web system designed to centralize and publicize events using modern *Front-End* development techniques. The platform was built with React, Context API, Styled-Components, and Ant Design, ensuring a responsive and efficient interface. Additionally, Firebase was used for authentication and data storage, optimizing development time. The adopted methodology involved requirements elicitation, prototyping, and system implementation, following best practices in *Front-End* development. The obtained results demonstrated the platform's effectiveness in organizing and presenting events, enhancing the user experience when searching for information. Future work includes implementing an administrative panel for better event management control.

Keywords: Web Development. Front-End. React. Event Promotion.

SUMÁRIO

1. INTRODUÇÃO

O avanço da tecnologia e da internet possibilitou a modernização e a otimização de processos que antes eram robustos, físicos e complexos, em métodos acessíveis, eficientes e tecnológicos. A sociedade, cada vez mais interligada ao universo digital, passou por diversas transformações ao longo do tempo, que consequentemente, refletem diretamente na maneira como projetos são concebidos e executados. Essa migração funcional pode ser observada, por exemplo, na organização de eventos (sejam eles acadêmicos, científicos, religiosos ou culturais), especialmente no ato de divulgação e inscrição; já que as *homepages* e os links de acesso podem substituir os panfletos e planilhas físicas anteriormente utilizados.

Nesse viés, considerando que um evento é uma ação planejada, organizada, coordenada e executada com o objetivo de atingir um público-alvo por meio da realização de um ato comemorativo, seja ele de finalidade mercadológica ou não (MATIAS, 2004), e que uma comunidade é um conjunto de pessoas que compartilham um espaço físico e estabelecem relações organizacionais e de influência mútua (COUTINHO, 2010), surge assim, a proposta do *CommunityEvents*. O projeto, desenvolvido no decorrer da disciplina de Estágio Supervisionado da Faculdade de Sistemas de Informação (FASI) da Universidade Federal do Pará (UFPA), Campus do Tocantins/Cametá, tem como objetivo centralizar e divulgar eventos em uma única plataforma. Dessa maneira, busca-se reduzir o esforço dos usuários na busca por eventos de seu interesse em diferentes fontes, oferecendo uma aplicação web que reúne todas as informações relevantes sobre cada evento. Sendo assim, optou-se por uma plataforma web, dado que, através da internet, as tecnologias da informação tornam-se um meio poderoso de comunicação, informação e realização de ações corporativas (VALENTE, 2014).

Portanto, este trabalho busca apresentar os conhecimentos adquiridos durante o estágio supervisionado no Laboratório de Abordagens de ensino Focadas no Aluno (LA FocA), vinculado à FASI. O foco principal é a aplicação de técnicas de desenvolvimento *Front-End*, com ênfase no uso da biblioteca React, para a criação da plataforma *CommunityEvents*. A utilização dessa tecnologia foi um diferencial na concepção, estruturação e organização da plataforma, tanto em relação aos dados quanto ao design; resultando em uma interface funcional, eficiente, acessível e responsiva, capaz de proporcionar uma experiência aprimorada ao usuário.

2. METODOLOGIA

A metodologia adotada durante o estágio para o desenvolvimento das atividades relacionadas à criação do projeto teve início com o estudo do desenvolvimento *Front-End* em React, por meio de cursos online disponibilizados aos integrantes do La FocA. Após a conclusão dos estudos e das práticas realizadas durante o curso, foi proposto, como projeto final do estágio, o desenvolvimento de uma Plataforma de Divulgação de Eventos. Para sua concepção, foram realizadas reuniões para a elicitação dos requisitos funcionais da plataforma, nos quais foram descritos os serviços que o sistema deve oferecer, detalhando as entradas específicas e seu comportamento em diferentes situações, especificando, assim, as funcionalidades que devem ser implementadas, conforme descrito em Sommerville (2011).

Concomitantemente ao levantamento de requisitos, decidiu-se pela utilização do Firebase, uma plataforma do Google que disponibiliza serviços de *Back-End*, permitindo uma maior dedicação e otimização do tempo na implementação do *Front-End* da plataforma. Dessa forma, foram configuradas as seguintes funcionalidades no Firebase:

- ***Authentication*:** disponibiliza o gerenciamento de acesso do usuário à aplicação, suportando vários métodos de autenticação como e-mail e senha.
- ***Firestore Database*:** banco de dados NoSQL baseado em coleções e documentos, permitindo o armazenamento de dados de forma escalável.
- ***Storage*:** armazena e gerencia arquivos, como imagens, vídeos e documentos.

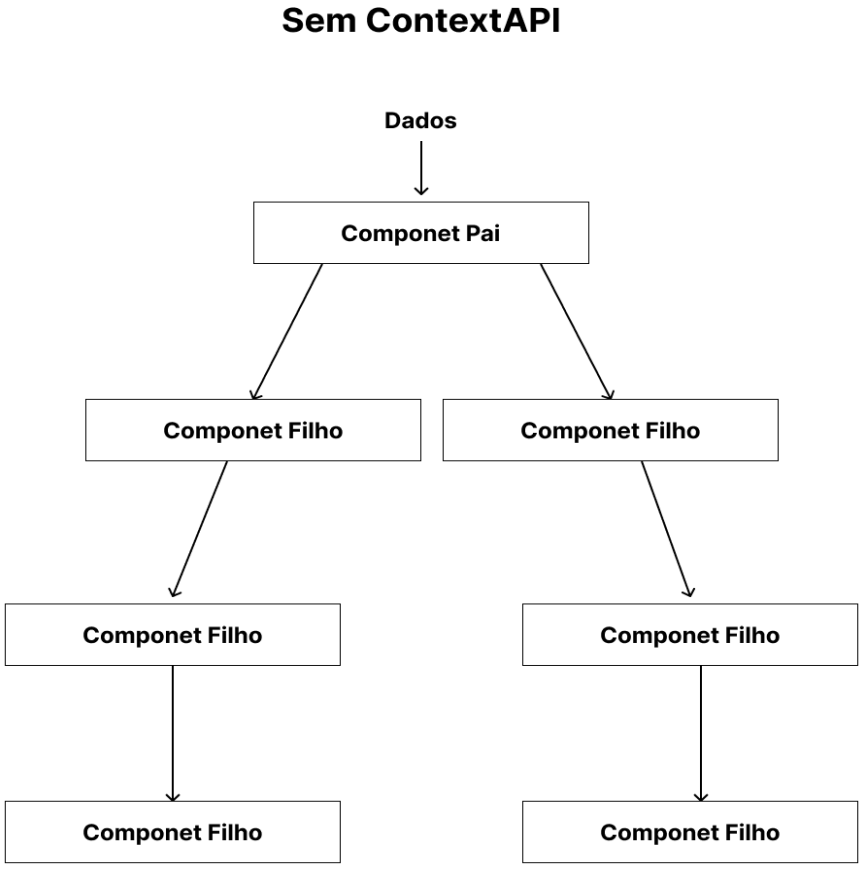
Após a configuração do Firebase, foram elaborados os protótipos da aplicação e, posteriormente, desenvolvido o design de interface da *CommunityEvents* no Figma. O material foi apresentado ao cliente (supervisor do estágio) para validação e possíveis ajustes. Com a aprovação do design, prosseguiu-se para a definição das tecnologias que seriam utilizadas no desenvolvimento do *Front-End*. Por fim, iniciou-se sua implementação permitindo a aplicação de técnicas no desenvolvimento da plataforma. As principais tecnologias e técnicas empregadas no projeto, e que serão abordadas ao longo deste trabalho, incluem: *React, Context API, Outlet* e *Ant Design* integrado ao *Styled-Components*.

3. TÉCNICAS UTILIZADAS NO FRONT-END DO PROJETO

3.1 CONTEXT API

Em uma aplicação React simples, a passagem de dados ocorre de forma unidirecional: de cima para baixo (do componente pai para o filho), utilizando *props*, conforme ilustrado na Figura 1. No entanto, esse método se torna inviável à medida que a aplicação cresce e a complexidade das interfaces aumentam. Com árvores de componentes cada vez mais profundas, a distância entre a raiz e suas folhas se torna maior, resultando em um código inchado e dificultando a escalabilidade da interface do usuário (UI – *User Interface*) (LE, 2021).

Figura 1 - Representação de uso sem *ContextAPI*



Fonte: Autor (2025)

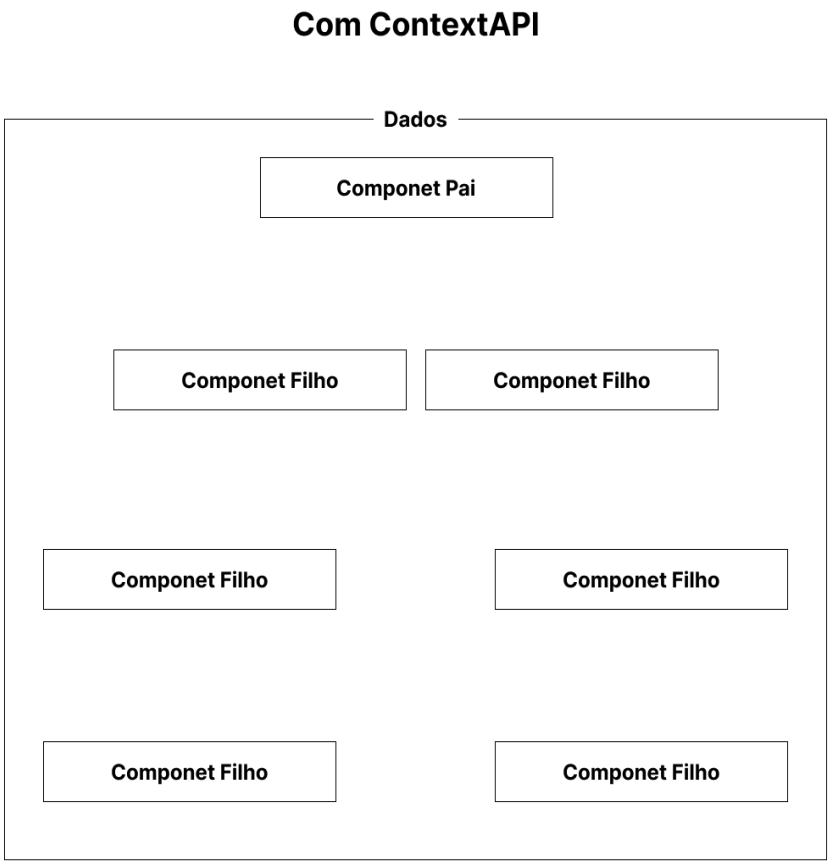
Esse problema é conhecido como *prop drilling*, no qual os dados de estado são passados manualmente por diversos componentes intermediários até alcançar aqueles que realmente necessitam deles. Essa situação pode ser comparada a uma viagem de trem, onde o passageiro (dado) percorre todas as estações (componentes), mas só desembarca no destino

final (componente que realmente precisa da informação). Esse processo torna o código mais complexo, difícil de manter e menos eficiente (LE, 2021).

Para evitar o problema conhecido como *prop drilling*, identificado no desenvolvimento do projeto *CommunityEvents*, optou-se pela utilização da biblioteca *Context API*, uma técnica proveniente do *React.js*, recomendada para o compartilhamento global de dados em aplicações web.

A Figura 2 ilustra o funcionamento desse recurso dentro de uma aplicação *React.js*. A principal diferença em relação à Figura 1 está na forma como os dados são acessados.

Figura 2 - Representação de uso com ContextAPI



Fonte: Autor (2025)

Com o uso da *Context API*, os dados podem ser compartilhados e acessados diretamente por qualquer componente que esteja dentro do *context provider*, sem a necessidade de repasse manual de propriedades entre os componentes intermediários. Essa abordagem contribui para um código mais modular e facilita a manutenção da aplicação.

Neste sentido, foi abordado ao projeto para a gestão do estado de filtros na plataforma *CommunityEvents*. Inicialmente, foi necessário definir duas interfaces para a

tipagem, as quais foram declaradas em um arquivo separado da função principal, conforme ilustrado na Figura 3.

Figura 3 - Tipos de dados que serão implementados no ContextAPI

```
export interface IContextFilter {
    handleSearch: string;
    selectEvent: string;
    selectCategory: string;
    type: string,
    category: string,
    setHandleSearch: (handleSearch: string) => void;
    setSelectEvent: (selectEvent: string) => void;
    setSelectCategory: (selectCategory: string) => void;
    setType: (type: string) => void;
    setCategory: (category: string) => void;
}

export interface IFilterProvider {
    children: React.ReactNode;
}
```

Fonte: Autor (2025)

A primeira interface, denominada *IContextFilter,* representa as variáveis e funções que compõem o estado global da aplicação, possibilitando o compartilhamento dessas informações entre os componentes. Já a interface *IFilterProvider* é responsável por definir a propriedade *children*, permitindo que qualquer componente válido do React seja encapsulado pelo provedor de contexto. Dessa forma, assegura-se a correta tipagem e o funcionamento adequado do provedor de estado no sistema.

Dadas as devidas implementações e definições de interfaces, foi desenvolvido o arquivo responsável por armazenar o contexto de filtro da plataforma. A Figura 4 ilustra a implementação do código em React utilizando *TypeScript*.

Figura 4 - Código de criação do *ContextAPI*

```tsx
import { createContext, useState } from "react";
import { IContextFilter, IFilterProvider } from "./types";

export const FilterContext = createContext<IContextFilter>({} as IContextFilter)

export function FilterProvider({ children }: IFilterProvider) {
    const [handleSearch, setHandleSearch] = useState<string>("")
    const [selectEvent, setSelectEvent] = useState<string>("Tipo evento")
    const [selectCategory, setSelectCategory] = useState<string>("Escolha uma categoria")
    const [type, setType] = useState<string>("")
    const [category, setCategory] = useState<string>("")

    return (
        <FilterContext.Provider value={{
            handleSearch,
            selectEvent,
            selectCategory,
            type,
            category,
            setHandleSearch,
            setSelectEvent,
            setSelectCategory,
            setType,
            setCategory,
        }}>
            { children }
        </FilterContext.Provider>
    )
}
```

Fonte: Autor (2025)

Inicialmente, realiza-se a importação dos módulos *createContext* e *useState*, bem como das tipagens previamente definidas e mencionadas anteriormente. O segundo passo consiste na criação do contexto, utilizando a função *createContext*, que é tipada com a interface *IContextFilter*. Em seguida, define-se a função responsável por armazenar as variáveis e estados que serão disponibilizados pelo *FilterContext.Provider*.

Por fim, foi criado um arquivo contendo a função *useFilter* (Figura 5), a qual encapsula a chamada do *useContext* e repassa o *FilterContext* definido na Figura 4. Essa abordagem evita a necessidade de múltiplas importações diretas do contexto ao longo da aplicação. Dessa forma, para acessar as variáveis globais, basta utilizar a função *useFilter*, conforme demonstrado na Figura 6.

Figura 5 - Função *useFilter*

```tsx
import { useContext } from "react";
import { FilterContext } from "../context/FilterProvider/index";

export function useFilter() {
    const context = useContext(FilterContext)

    return context
}
```

Fonte: Autor (2025)

Figura 6 - Utilização do *useFilter*

```
const {type, category, handleSearch} = useFilter()
```

Fonte: Autor (2025)

3.2 NAVEGAÇÃO

Para compreender o conceito de Outlet e seu impacto no desenvolvimento da *CommunityEvents*, é essencial conhecer o *React Router Dom*. Essa biblioteca do *React* fornece ferramentas para o gerenciamento da navegação em aplicações web, simplificando o controle das URLs e do estado da aplicação.

Com o *React Router Dom*, é possível definir padrões de URL e especificar quais componentes de interface devem ser renderizados para cada rota (SIMPLILEARN, 2024). Dessa forma, a navegação torna-se dinâmica e eficiente, garantindo uma melhor experiência ao usuário.

A Figura 7 apresenta a implementação da estrutura de rotas na aplicação *CommunityEvents*, desenvolvida em *React*.

Figura 7 - Sistema de Rotas da *CommunityEvents*

```jsx
import { createBrowserRouter } from "react-router-dom";
import { Home } from "../pages/Layout/Home";
import { Events } from "../pages/Layout/Events";
import { About } from "../pages/Layout/About";
import { Login } from "../pages/Layout/Login";
import { Register } from "../pages/Layout/Register";
import { Panel } from "../pages/Dashboard/Panel";
import { EventsDash } from "../pages/Dashboard/Events";
import { CreateEvent } from "../pages/Dashboard/CreateEvent";
import { UpdateEvent } from "../pages/Dashboard/UpdateEvent";
import { Event } from "../pages/Layout/Event"
import { HomeDash } from "../pages/Dashboard/HomeDash";
import { Profile } from "../pages/Dashboard/Profile";

export const router = createBrowserRouter([
    {
      path: "/",
      element: <Home />,
    },
    {
      path: "/events",
      element: <Events />
    },
    {
      path: "/about",
      element: <About />
    },
    {
      path: "/login",
      element: <Login />
    },
    {
      path: "/register",
      element: <Register />
    },
    {
      path: "/dashboard",
      element: <Panel />,
      children:[
        {
          path: "/dashboard/",
          element: <HomeDash />
        },
        {
          path: "/dashboard/profile/:id",
          element: <Profile />
        },
        {
          path: "/dashboard/events",
          element: <EventsDash />
        },
        {
          path: "/dashboard/events/create",
          element: <CreateEvent />
        },
        {
          path: "/dashboard/events/update/:id",
          element: <UpdateEvent />
        }
      ]
    },
    {
      path: "/event/:id",
      element: <Event />
    }
  ]);
```

Fonte: Autor (2025)

A variável denominada *router* recebe uma função da biblioteca *React Router Dom*, permitindo a criação da estrutura de rotas da plataforma, conforme ilustrado na Figura 7. Essa estrutura é baseada em um array de objetos, onde cada objeto contém as seguintes propriedades:

- **path**: representa o caminho da URL correspondente à rota.
- **element**: define o componente de interface (UI) a ser renderizado para a respectiva rota.

No entanto, quando o path recebe a rota "/**dashboard**", onde será implementada a técnica do *Outlet*, torna-se necessário incluir a propriedade *children*. Essa propriedade é responsável por armazenar as rotas filhas do *Dashboard*, seguindo a arquitetura *Single Page Application* (SPA). Essa abordagem permite a renderização apenas dos componentes necessários, evitando o recarregamento completo da página e proporcionando uma experiência mais fluida ao usuário.

Além disso, para tornar as rotas acessíveis em toda a aplicação, foi necessário importar o arquivo *router* e passá-lo para o *RouterProvider*. Esse provedor tem a função de gerenciar a navegação entre os componentes previamente implementados.

Por fim, o App é exportado com os respectivos arquivos apresentados na Figura 8.

Figura 8 - Função App

```
import { RouterProvider } from "react-router-dom"
import { router } from "./routes"

function App() {

  return (
    <RouterProvider router={router} />
  )
}

export default App
```

Fonte: Autor (2025)

3.2.1 Outlet

Após a implementação de toda a arquitetura de navegação, foi aplicada a técnica do *Outlet* no projeto, mais especificamente no *Dashboard* do usuário. Segundo React Router Documentation (2025), o uso do *Outlet* é uma prática recomendada para gerenciar a renderização de componentes em estruturas de navegação complexas, como em *dashboards*

ou sistemas com múltiplas seções interativas. Esse recurso é fundamental para o gerenciamento de *layouts* complexos, permitindo que a estrutura principal da página permaneça consistente, enquanto apenas determinados componentes são atualizados dinamicamente conforme as rotas definidas na propriedade *children*, conforme ilustrado na Figura 7.

A utilização do Outlet no código do projeto ocorreu conforme mostra a Figura 9.

Figura 9 - Utilização do Outlet

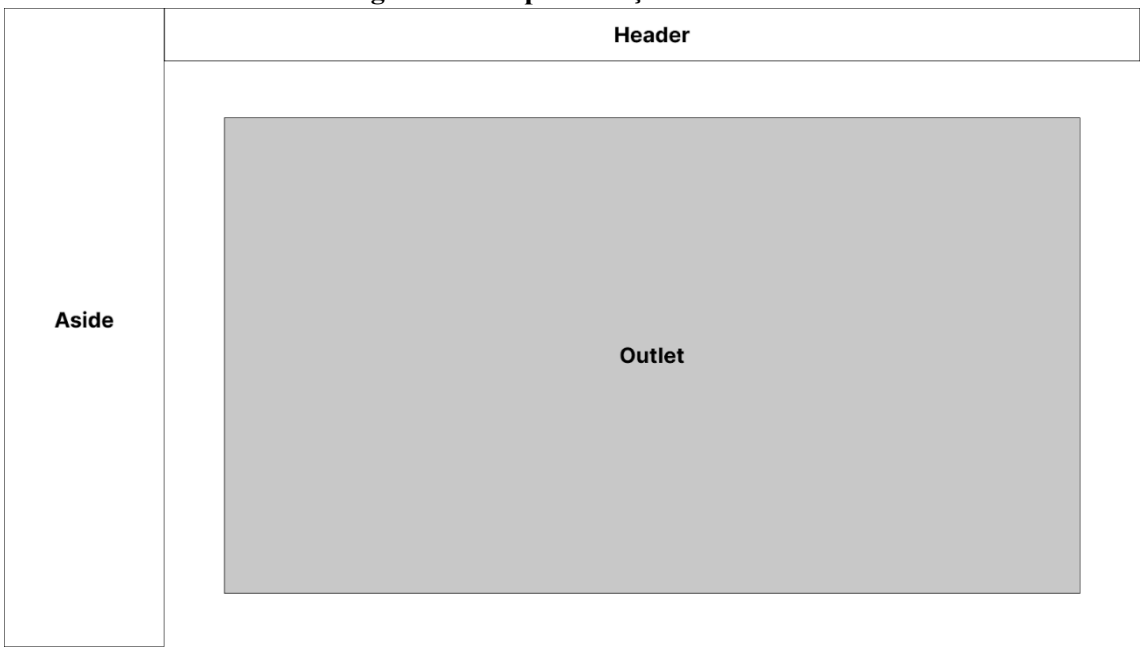
```tsx
export function Panel() {
    const auth = useAuth()
    const { getEvent } = useEvent()
    const [active, setActive] = useState(false)
    const navigate = useNavigate()
    const { activeModal, activeLogout } = useModal()
    const [userImage, setUserImage] = useState("")
    const [nameUser, setNameUser] = useState("")

    if(!auth?.user) {···
    }

    async function getUser() {···
    }

    getUser()

    return (
        <LayoutAnt>
          <SiderAnt trigger={null}>···
          </SiderAnt>
          <LayoutAnt>
            <HeaderAnt>
              <h3>Dashboard</h3>
              <div className="profile">
                <div className="img_profile">
                  <img src={userImage ? userImage : ""} alt="" />
                </div>

                <div className="info_user">
                  <p>{nameUser}</p>
                </div>

                <button className='active_options' onClick={() => setActive(!active)}>
                  <DownOutlined />
                </button>

                {active && <OptionsProfile user={auth?.user} setActive={setActive}/>}
              </div>
            </HeaderAnt>

            <ContentAnt>

              <Outlet />

            </ContentAnt>

            {activeLogout && <Modal />}
            {activeModal && <ModalTrash />}
          </LayoutAnt>
        </LayoutAnt>
      );
}
```

Fonte: Autor (2025)

O *outlet* é inserido dentro do componente denominado *Panel*. Este componente serve como a estrutura base do *Dashboard*, onde ficam os elementos fixos da interface, como cabeçalhos e barras laterais. Ao posicionar o *Outlet* dentro da estrutura *ContentAnt*, conforme

a Figura 10, ele se torna o espaço reservado para a renderização dinâmica do conteúdo, seguindo as rotas definidas na Figura 7.

Figura 10 - Representação do Outlet



Fonte: Autor (2025)

Conforme ilustrado, os componentes *Aside* e *Header* são elementos estáticos que permanecem visíveis em todas as páginas do *Dashboard*. Já a área destacada em cinza representada na Figura 10 evidencia a seção dinâmica, que será atualizada conforme a rota especificada. Esse método garante uma navegação eficiente e fluida, alinhada ao conceito de SPA.

3.3 STYLED-COMPONENTS E ANT DESIGN

Para a criação e estruturação de cores, design de páginas e estilos, optou-se pela utilização do *CSS* em forma de componente, especificamente por meio da tecnologia *Styled-Components*. Esta biblioteca permite a escrita de *Cascading Style Sheets* (CSS) diretamente no *JavaScript*, proporcionando um controle mais preciso sobre os estilos aplicados aos componentes.

Ao adotar essa abordagem, é possível otimizar o desempenho da aplicação, reduzindo o volume de código carregado na página e garantindo que cada componente tenha seus estilos encapsulados. Uma das principais vantagens do *Styled-Components* é a eliminação da duplicação de nomes de classes, uma vez que a tecnologia gera automaticamente identificadores únicos para cada estilo aplicado. Dessa forma, evita-se a repetição de classes globais e conflitos de estilos que poderiam ocorrer em abordagens

tradicionais com *CSS* puro ou pré-processadores. Além disso, como os estilos são vinculados diretamente aos componentes, a manutenção do código se torna mais eficiente e organizada.

Figura 11 - Criação de um Componente de estilo

```
export const GlobalLayoutButton = styled.a`
    padding: 12px 18px;
    font-size: 14px;
    font-weight: 600;
    color: white;
    background-color: #2d17af;
    border-radius: 8px;

    &:hover {
        background-color: var(--button-hover);
    }
`
```

a) Código **b)** Resultado visual

Fonte: Autor (2025)

A Figura 11a ilustra de forma simplificada a criação de um componente utilizando *Styled-Components*. No código, uma variável é declarada para receber a biblioteca *styled*, seguida do método “a”, indicando que o componente será estilizado como um botão mas não deixando de ser um link, redirecionando o usuário para a página de registro. Já na Figura 11b, é possível visualizar o botão aplicado com as devidas estilizações. Um ponto importante a ser destacado é que esse componente foi projetado para uso global na *Home Page* da plataforma, garantindo consistência visual e reutilização eficiente ao longo do projeto. Essa abordagem reforça a modularidade e a escalabilidade do código, permitindo que o botão seja facilmente adaptado e mantido sem a necessidade de replicação de estilos.

No desenvolvimento dos estilos da plataforma, especialmente no *dashboard* do usuário, optou-se pela utilização de componentes prontos da biblioteca *Ant Design*. Essa escolha se justifica pela facilidade de uso e pela agilidade no desenvolvimento, já que a biblioteca oferece estruturas e elementos pré-definidos, garantindo um design consistente e uma implementação mais eficiente.

Entretanto, como os componentes do *Ant Design* possuem estilos padrão, foi necessário aplicar a técnica de herança do *Styled-Components* para personalizá-los conforme as necessidades do projeto. Essa abordagem permitiu modificar a aparência dos componentes sem comprometer suas funcionalidades nativas. A Figura 12 explora essa técnica em mais detalhes, demonstrando sua aplicação prática na customização da interface.

Figura 12 - Conceito de herança abordado no *Styled-Components*

```
import styled from "styled-components";
import { Input } from "antd"

export const Search = styled(Input.Search)`
    .ant-input-outlined:focus {
        box-shadow: none;
    }
    .ant-input-group > .ant-input-group-addon:last-child
    .ant-input-search-button:not(.ant-btn-primary):hover {
        color: #2d17af;
    }
    .ant-btn-variant-outlined:not(:disabled):not(.ant-btn-disabled):hover {
        color: #2d17af;
        border-color: #2d17af;
    }
    .ant-input:focus + .ant-input-group-addon
    .ant-input-search-button:not(.ant-btn-primary) {
        border-inline-start-color: #2d17af;
    }
`
```

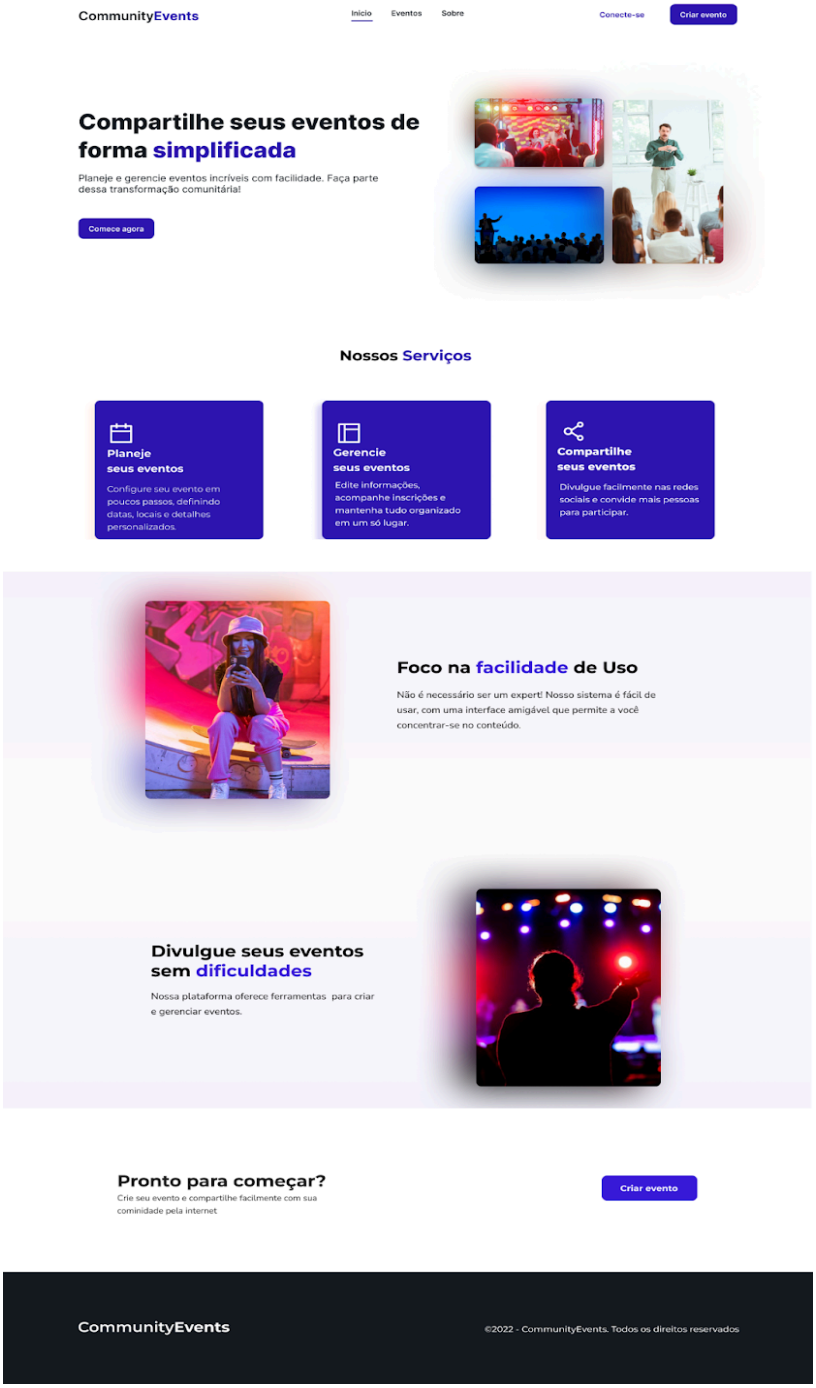
Fonte: Autor (2025)

Para herdar componentes externos como o *Ant Design* no *Styled-Components*, basta apenas importar o elemento desejado e passar para o método *styled*. Com isso, o componente importado já pode ser estilizado, mas é necessário verificar as classes padrões para uma estilização mais profunda como foi explorado na Figura 12.

4. PLATAFORMA COMMUNITY EVENTS

A Figura 13 representa a página inicial da plataforma *CommunityEvents*. Nessa página, foram aplicadas técnicas de estilização componentizada, garantindo uma estrutura modular e reutilizável. Um dos principais exemplos dessa abordagem é a reutilização do componente de botão em três seções diferentes, proporcionando consistência visual e facilidade de manutenção do código. O principal objetivo da página inicial é apresentar os serviços oferecidos pela plataforma de forma clara e intuitiva.

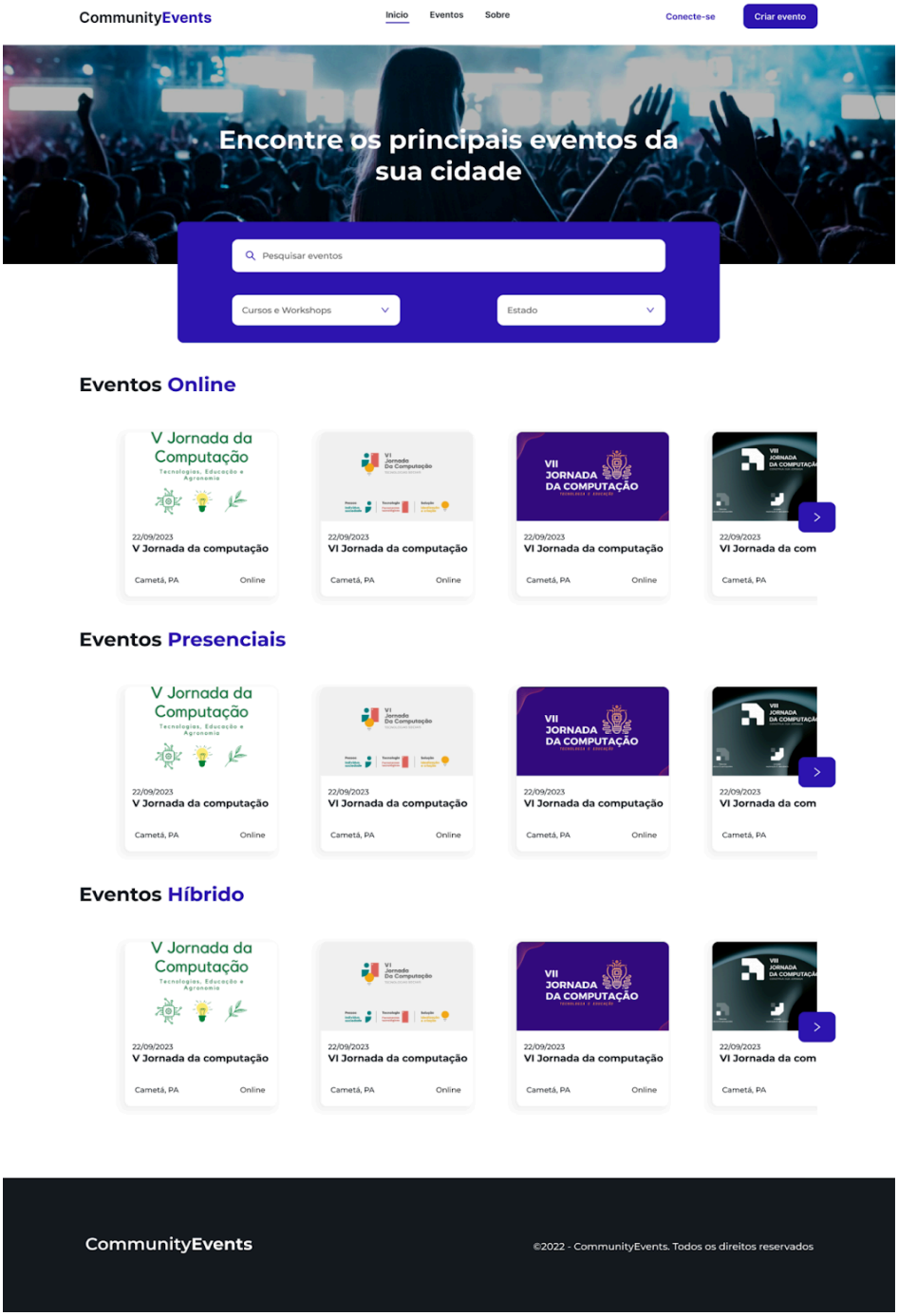
Figura 13 - Página de início da *CommunityEvents*



Fonte: Autor (2025)

A Figura 14 tem como principal objetivo exibir todos os eventos cadastrados na plataforma. Para facilitar a navegação, os eventos são apresentados separadamente de acordo com seu tipo, permitindo que os usuários identifiquem rapidamente as categorias disponíveis.

Figura 14 - Página de Eventos da *CommunityEvents*



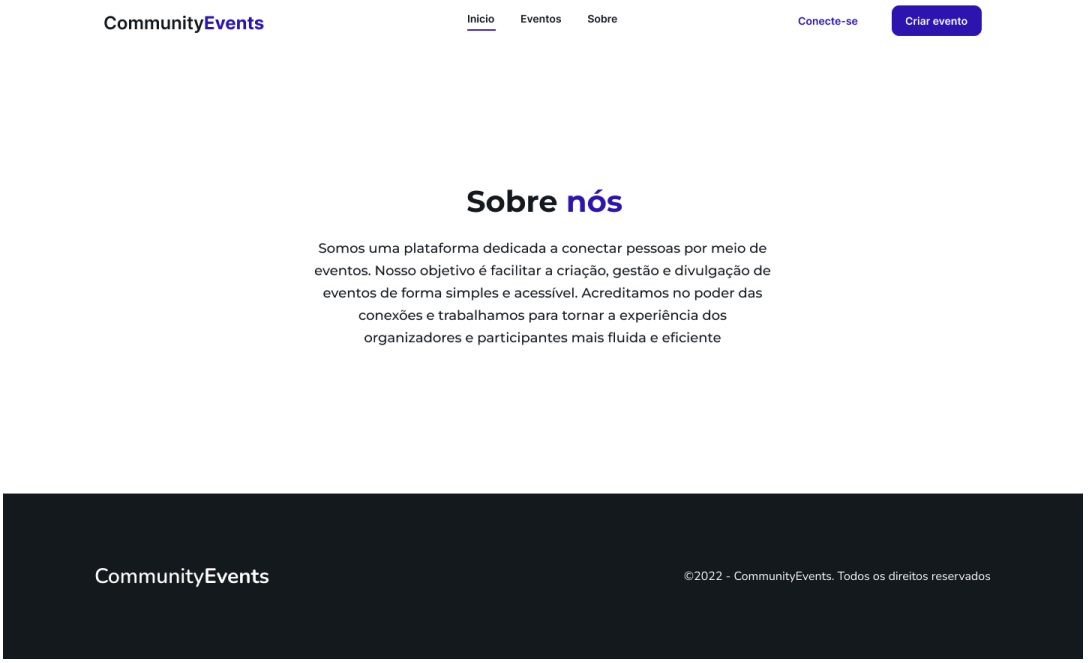
Fonte: Autor (2025)

Além disso, foi aplicada a técnica do *Context API* para gerenciar o estado global da aplicação, possibilitando que o usuário filtre os eventos de acordo com suas preferências. Essa

abordagem melhora a performance e a experiência do usuário, garantindo uma interação dinâmica e fluida com a plataforma.

A página "Sobre", representada na Figura 15, apresenta uma estrutura relativamente simples, focada em fornecer informações essenciais sobre a plataforma. Para otimizar o desenvolvimento e manter a consistência visual, foram reaproveitados componentes já criados na página inicial, aplicando o conceito de reutilização com *Styled Components*.

Figura 15 - Página Sobre



Fonte: Autor (2025)

A página representada na Figura 16 é dedicada exclusivamente à exibição de informações detalhadas sobre um evento de interesse do usuário. Seu objetivo é fornecer uma visão prévia do evento, apresentando dados relevantes de forma organizada e intuitiva.

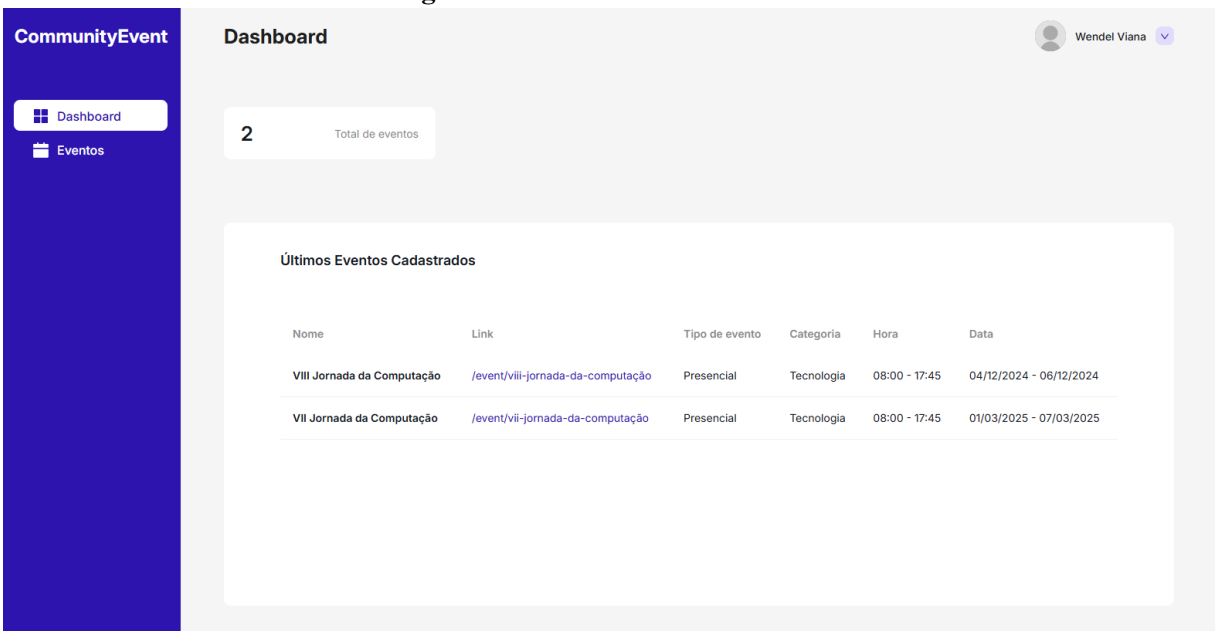
Entre as informações exibidas na página, estão: título do evento, data de início e término, horário de início e fim, descrição detalhada e um link oficial que redireciona o usuário para a página de origem do evento. Essa estrutura facilita a compreensão do conteúdo e permite que o usuário acesse rapidamente mais detalhes ou efetue sua inscrição diretamente na plataforma de origem.

Após realizar o login na plataforma, o usuário terá acesso à página *Dashboard* (Figura 17), que apresenta algumas informações. Nessa página, o usuário poderá visualizar o total de eventos cadastrados, bem como uma lista dos últimos eventos adicionados.

Figura 16 - Página dedicada para apresentar informações prévias do evento

Fonte: Autor (2025)

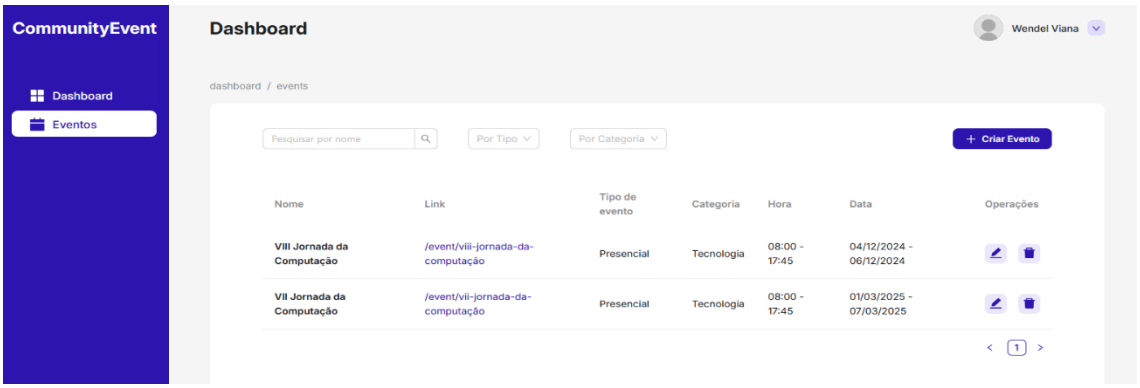
Figura 17 - *Dashboard* do usuário



Fonte: Autor (2025)

Além disso, a interface inclui elementos que melhoram a navegação e a experiência do usuário. No canto superior esquerdo, um indicador exibe a página atual em que o usuário se encontra, facilitando a orientação dentro da plataforma. Já no canto superior direito (Figura 18), são exibidas informações básicas do usuário, como sua foto de perfil e nome, proporcionando uma experiência mais personalizada e intuitiva.

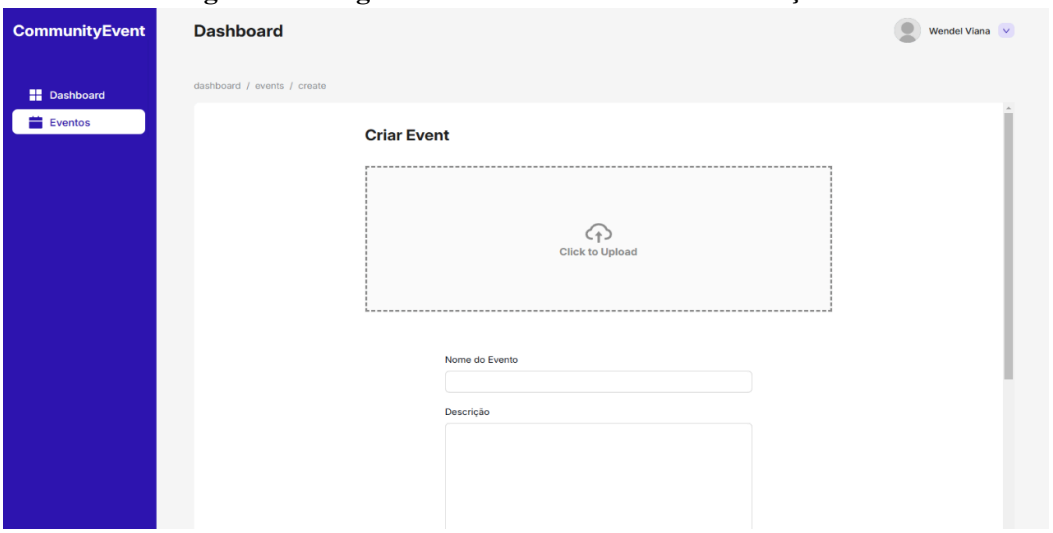
Figura 18 - Página de eventos cadastrados pelo usuário



Fonte: Autor (2025)

A página representada na Figura 19 oferece ao usuário um conjunto de funcionalidades essenciais para a gestão de eventos na plataforma. Nela, é possível realizar o cadastro de novos eventos, fazer o upload de arquivos relacionados e excluir eventos já registrados. Além dessas funcionalidades, a página também conta com uma opção de filtragem, permitindo que o usuário encontre eventos específicos de maneira rápida.

Figura 19 - Página onde fica o formulário de criação de eventos

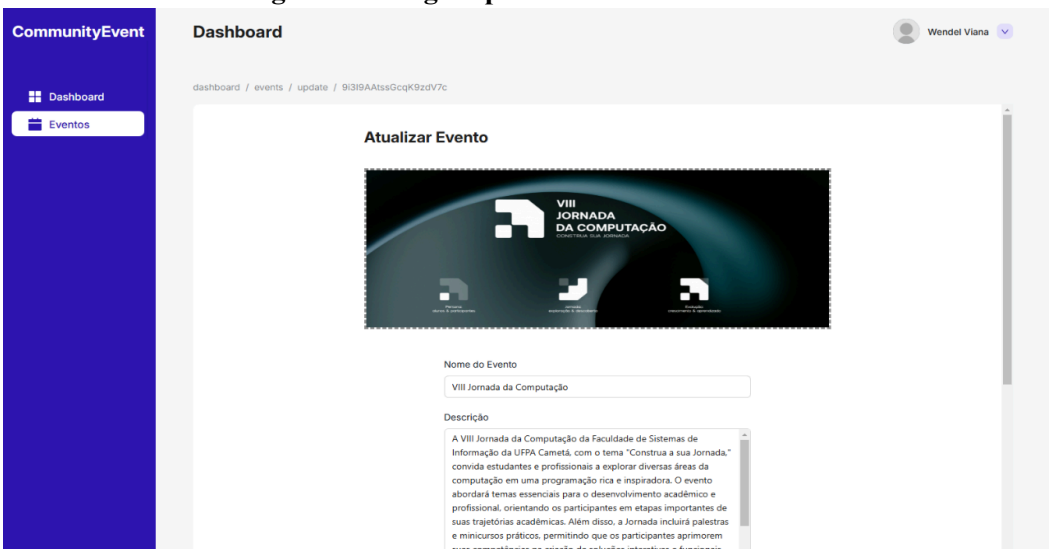


Fonte: Autor (2025)

Nesta rota, o usuário pode criar um evento preenchendo um formulário com todas as informações necessárias. Os dados solicitados incluem: imagem do evento, nome, descrição, data de início e término, horário de início e fim, tipo de evento, categoria, local e o link original para a página oficial do evento.

Na Figura 20, ao clicar no ícone de edição apresentado na Figura 18, o usuário tem a possibilidade de atualizar as informações do evento selecionado. A página de edição segue o mesmo padrão visual e estrutural do formulário de criação de eventos, garantindo uma experiência consistente dentro da plataforma. A principal diferença é que, na página de edição, os dados do evento já são carregados automaticamente nos campos do formulário, permitindo que o usuário faça as alterações necessárias. Essa abordagem facilita a atualização das informações sem a necessidade de inserir todos os dados novamente.

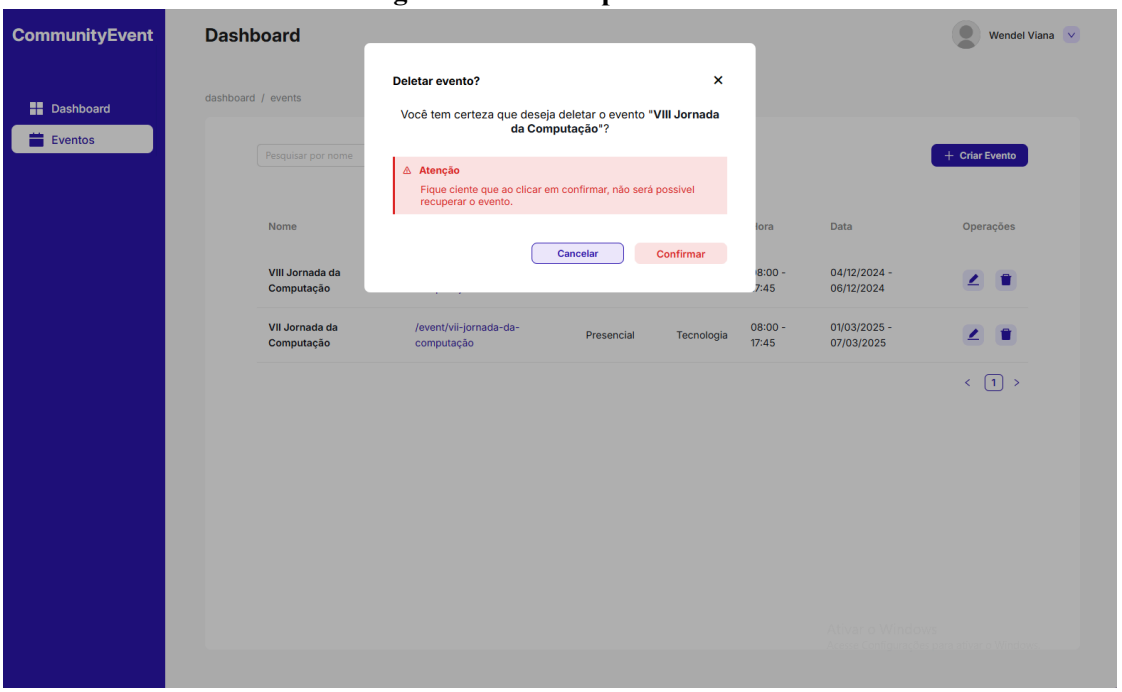
Figura 20 - Página para atualizar um Evento



Fonte: Autor (2025)

O *Modal* de exclusão da Figura 21 é exibido quando o usuário clica no ícone de lixeira localizado na linha do respectivo evento. Ao ser renderizado, o *Modal* apresenta informações sobre o evento que será excluído, além de exibir alertas sobre as possíveis consequências da remoção. Para garantir uma decisão consciente, o usuário terá duas opções dentro do *Modal*: cancelar a ação ou confirmar a exclusão de forma definitiva.

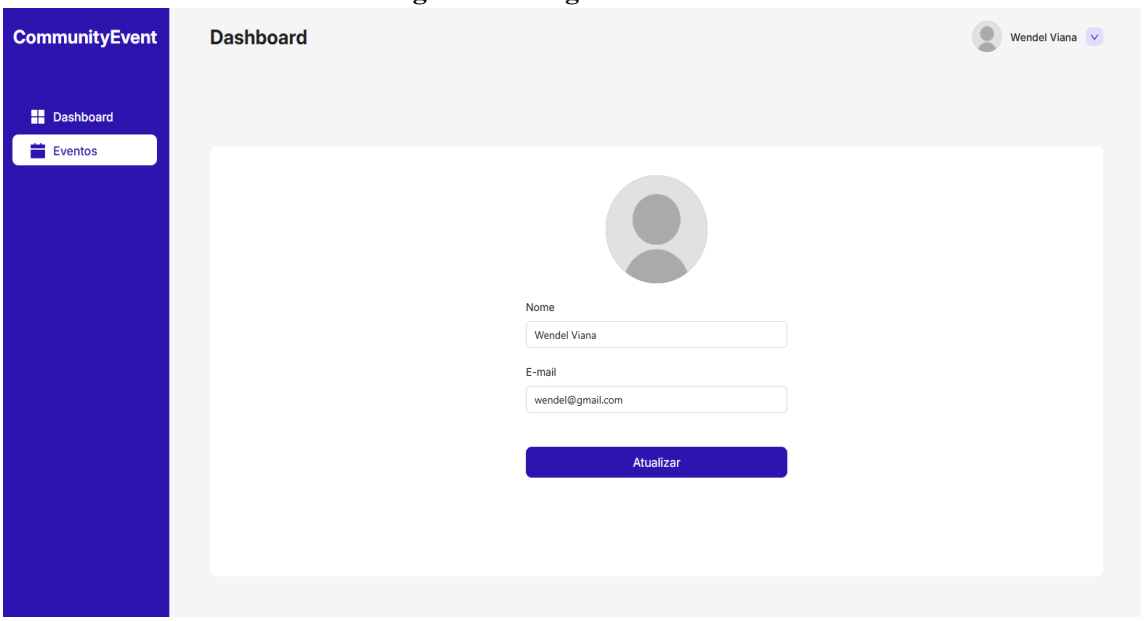
Figura 21 - Modal para excluir evento



Fonte: Autor (2025)

Nessa página da Figura 22, o usuário vai poder verificar seu nome e email cadastrado e se for de seu desejo atualizá-los.

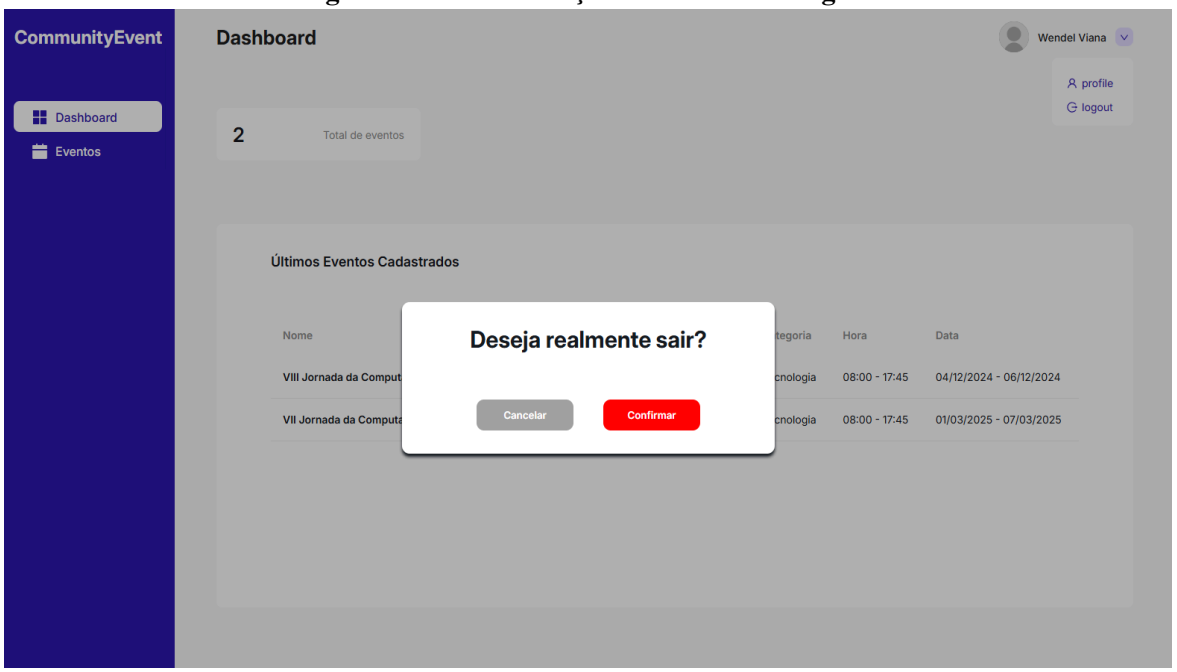
Figura 22 - Página de usuário



Fonte: Autor (2025)

Após o usuário clicar na parte superior do lado direito e escolher a opção logout será renderizado o modal para confirmação da ação, conforme Figura 23. O usuário vai poder escolher se realmente quer deslogar da página ou cancelar a ação.

Figura 23 - Renderização do Modal de Logout



Fonte: Autor (2025)

5. CONSIDERAÇÕES FINAIS

Este projeto teve como objetivo aplicar os conhecimentos adquiridos no estágio supervisionado realizado no grupo LA FocA, com ênfase no desenvolvimento *Front-End*. Durante o estágio, recebi apoio por meio de cursos da plataforma Udemy, que abordam não apenas programação, mas também áreas como design e marketing.

Os conhecimentos adquiridos foram aplicados no desenvolvimento da plataforma *CommunityEvents*, onde busquei utilizar as melhores práticas de *Front-End* para criar códigos eficientes, de fácil compreensão e manutenção. Esse estágio permitiu desenvolver e validar minhas habilidades para atuar na área de programação web, mais especificamente na função de desenvolvedor *Front-End*.

Como próximos passos, vislumbro a criação de uma arquitetura ainda mais robusta, além da implementação de um painel de controle com novas funcionalidades. Em particular, destaco a necessidade de incluir um controle Admin, que, devido ao tempo limitado de desenvolvimento, não foi implementado na versão atual. Este controle seria essencial para gerenciar e monitorar o conteúdo exibido na plataforma, evitando a criação ou exploração de conteúdos sensíveis ou indesejados. Além disso, a manutenção contínua da plataforma será fundamental para garantir sua estabilidade e a resolução de eventuais *bugs*.

REFERÊNCIAS

COUTINHO, HELEN RITA MENEZES. **Organização de eventos: curso técnico em hospedagem.** 1ª ed. Manaus : Centro de Educação Tecnológica do Amazonas, 2010.

LE, Thanh. Comparison Of State Management Solutions Between Context Api And Redux Hook In Reactjs. 2021.

LINS, Gabriel de Souza. **Utilizando ReactJS para o desenvolvimento de um sistema de: alocação e reserva de salas no campus da UFC em Quixadá**. 2019.

MATIAS, Marlene. **Organização de eventos: procedimentos e técnicas.** 3ª ed. Barueri: Manole, 2004.

REACT ROUTER https://reactrouter.com/

SIMPLILEARN. **Por que você deve usar um roteador no React.js?** 2024. Disponível em: https://www.simplilearn.com/tutorials/reactjs-tutorial/routing-in-reactjs#:~:text=BootcampExplore%20Program-,What%20Is%20a%20React%20Router%3F,create%20complicated%20client%2Dside%20apps. Acesso em:

SOMMERVILLE, Ian. Software engineering (ed.). **America: Pearson Education Inc**, 2011.

VALENTE, José Armando. A comunicação e a educação baseada no uso das tecnologias digitais de informação e comunicação. **UNIFESO-Humanas e Sociais**, v. 1, n. 01, p. 141-166, 2014.