FACULDADE DE TECNOLOGIA, CIÊNCIAS E EDUCAÇÃO

Graduação

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Desenvolvimento de Plataforma Web e Mobile para Descoberta e Organização de Eventos Locais

Augusto Iseppe Balan
Matheus Vinicius Silva Denofrio da Cruz

Orientador: Alessandro Viola Pizzoleto

RESUMO

O panorama atual do gerenciamento e da divulgação de eventos exige uma arquitetura de sistema robusta e integrada, capaz de sincronizar plataformas móveis e *web*. Este trabalho descreve o desenvolvimento de uma solução completa e escalável para a gestão de eventos, estruturada em um aplicativo móvel nativo e um sistema *web* administrativo centralizado. O objetivo primordial foi desenvolver uma plataforma modular que otimize a listagem, filtragem e o gerenciamento de eventos em tempo real, visando a melhoria da experiência do usuário e a eficiência operacional. A arquitetura do sistema é dividida em duas camadas complementares. O aplicativo móvel, desenvolvido em *Flutter*, empregou o padrão MVVM *(Model-View-ViewModel)* em sinergia com o BLoC (*Business Logic Component*) para o gerenciamento de estado. Esta abordagem garante a separação de

responsabilidades, facilita a manutenção e suporta a escalabilidade da plataforma, oferecendo funcionalidades como listagem dinâmica, filtros por categoria e data, visualização detalhada e marcação de favoritos. A gestão administrativa é realizada por um painel web implementado com PHP e MySQL. Este sistema garante o CRUD (Criação, Leitura, Atualização e Exclusão) completo de eventos, anunciantes e locais, sendo reforçado por um robusto controle de acesso baseado em papéis e permissões. O sistema *web* atua como o *backend* de gestão centralizada, fornecendo a base de dados para futuras integrações. Os resultados incluem protótipos funcionais que validam a arquitetura modular e responsiva em ambas as plataformas. Conclui-se que o sistema é viável para a gestão de eventos, com recomendações para trabalhos futuros que envolvem a implementação de notificações *push* e serviços de geolocalização para ampliar o engajamento do usuário.

Abstract

The current landscape of event management and dissemination demands a robust and integrated system architecture capable of synchronizing mobile and web platforms. This work presents the development of a complete and scalable solution for event management, composed of a native mobile application and a centralized web-based administrative system. The main objective was to develop a modular platform that optimizes real-time event listing, filtering, and management, aiming to enhance user experience and operational efficiency.

The system architecture is structured into two complementary layers. The mobile application, developed with Flutter, employs the MVVM (Model-View-ViewModel) pattern in conjunction with the BLoC (Business Logic Component) for state management. This approach ensures clear separation of concerns, facilitates maintenance, and supports platform scalability, offering features such as dynamic listings, category and date filters, detailed views, and favorites management.

Administrative management is carried out through a web panel implemented in PHP and MySQL. This system provides full CRUD (Create, Read, Update, and Delete) operations for events, advertisers, and venues, reinforced by a robust role- and permission-based access control. The web system functions as the centralized backend, supplying the database infrastructure for future integrations.

The results include functional prototypes that validate the modular and responsive architecture across both platforms. It is concluded that the system is viable for event management, with recommendations for future work such as implementing push notifications and geolocation services to enhance user engagement.

Palavras-chave: Flutter, MVVM, BLoC, PHP, MySQL, Tailwind, HTML, CSS, JavaScript, eventos, mobile, web.

1. Introdução

Os eventos culturais, sociais, educacionais e de lazer fazem parte do cotidiano das pessoas, tanto em áreas urbanas quanto em rurais, sendo elementos essenciais para a promoção do convívio social, o fortalecimento da economia local e o enriquecimento cultural de uma comunidade. Atualmente, observa-se uma grande diversidade de eventos, que variam desde feiras de pequenos produtores e *workshops* até *shows*, festivais e celebrações comunitárias. Entretanto, a fragmentação das informações sobre esses eventos representa um desafio significativo tanto para organizadores, que enfrentam dificuldades na divulgação de suas atividades/eventos, quanto para o público, que muitas vezes desconhece as opções de lazer disponíveis em sua região (MELO; RIBEIRO, 2020).

Embora existam diversas plataformas de gerenciamento e divulgação de eventos, como *Eventbrite* e *Sympla*, a maioria delas não oferece a granularidade e o foco necessários para promover eventos estritamente locais ou de menor escala. Por exemplo, informações sobre *happy hours*, apresentações de artistas regionais e eventos em pequenos espaços ainda permanecem restritas a grupos de redes sociais, aplicativos de mensagens ou meios físicos de divulgação, como panfletos. Essa dispersão informacional limita a visibilidade de iniciativas culturais relevantes e reduz o potencial de engajamento do público, resultando em menor participação e perda de oportunidades para a comunidade.

É nesse contexto de dispersão e ineficiência comunicacional que se insere o projeto "Vamos Sair Hoje", proposto como uma solução tecnológica *web/mobile* unificada tanto para o gerenciamento quanto para a descoberta de eventos locais. Este trabalho propõe o desenvolvimento de uma plataforma *web* e um aplicativo *mobile*, no qual o sistema *web* utiliza uma interface responsiva, composta por uma interface pública para descoberta de eventos e um painel administrativo para gestão de conteúdo, ambos compartilhando a mesma base de dados MySQL. A plataforma *web* oferece funcionalidades administrativas completas de *CRUD* (Criação, Leitura, Atualização e Exclusão) de eventos, sistema de autenticação de usuários com diferentes níveis de permissão, e uma interface pública otimizada para descoberta e visualização de eventos em tempo real.

O sistema prioriza funcionalidades intuitivas de cadastro de eventos, filtros de busca por localização, data e categoria, além de ferramentas de divulgação eficientes com sistema de auditoria.

O aplicativo integra e complementa o sistema de gerenciamento de eventos, trazendo uma sinergia de plataformas, gerenciando e divulgando eventos. O aplicativo é uma ponte entre organizadores e público, servindo principalmente para o acesso e a descoberta de eventos locais, devido ao seu alto alcance, acessibilidade e mobilidade.

Dessa forma, o objetivo geral deste trabalho é desenvolver, implementar e validar um sistema *web* e *mobile* para descoberta, organização e divulgação de

eventos locais, facilitando a conexão ágil e eficiente entre organizadores de eventos e o público interessado.

2. Justificativa

A relevância deste estudo decorre tanto de sua contribuição prática, ao propor uma solução tecnológica *web/mobile* de impacto social para o fortalecimento de comunidades locais, quanto de sua contribuição acadêmica, ao aplicar e validar o uso de tecnologias *web* consolidadas (*PHP, MySQL, Tailwind CSS*) e *mobile* (*Flutter*) no desenvolvimento de sistemas de alta performance e acessibilidade. A utilização do *PHP* justifica-se por sua maturidade, ampla documentação e facilidade de hospedagem, enquanto o *MySQL* oferece robustez e confiabilidade para o gerenciamento de dados relacionais complexos. A opção pelo uso do *Flutter* se justifica pelo desenvolvimento *Cross-Platform* (múltiplas plataformas), que com uma única base de código, teremos aplicativos para *Android* e *IOS*, tendo um baixo custo, consistência, plataforma nativa e facilidade de construção e interfaces complexas.

Além disso, o sistema proposto busca resolver um problema real e recorrente nas comunidades — a dispersão das informações sobre eventos — ao oferecer uma interface *web* moderna, responsiva e acessível que centraliza todas as etapas do gerenciamento e da descoberta de eventos, juntamente com o dispositivo *mobile*, alavancando ainda mais a divulgação desses eventos. A iniciativa também contribui para a democratização do acesso à informação cultural e para o incentivo à economia criativa, especialmente em contextos pós-pandêmicos, nos quais o fortalecimento de vínculos sociais e culturais é essencial.

3. Revisão Bibliográfica

A seguir são apresentados os fundamentos teóricos que embasam o desenvolvimento da plataforma proposta. A revisão abrange conceitos relacionados às plataformas digitais de eventos, ao funcionamento da economia de plataformas e ao cenário atual de dispersão de informações, destacando a necessidade de centralização para facilitar o acesso dos usuários. Essas referências fornecem o suporte conceitual necessário para compreender o problema abordado e orientar as escolhas metodológicas e tecnológicas adotadas no projeto.

3.1. Plataformas de Eventos, Economia de Plataforma Digital o Contexto da Dispersão de Eventos e a Necessidade de Centralização

A transformação digital tem impactado significativamente a forma como eventos são organizados, divulgados e consumidos através de plataformas *web* ou dispositivos *mobile*. O advento dessas ferramentas possibilitou a descentralização da informação e a ampliação do alcance de atividades culturais, sociais e comerciais

(PARKER et al., 2016). Essas plataformas atuam como intermediárias entre produtores e consumidores, promovendo a chamada economia de plataforma digital.

No contexto dos eventos locais, observa-se uma lacuna entre as grandes soluções globais — como *Eventbrite* e *Sympla* — e as necessidades regionais de pequenos produtores e comunidades. O projeto "Vamos Sair Hoje" se insere nesse cenário como uma proposta inovadora, ao oferecer uma infraestrutura *web* e *mobile* unificada que conecta a gestão administrativa à descoberta e consumo de eventos, atendendo a uma lacuna concreta na realidade das comunidades locais.

O fenômeno da Economia de Plataforma Digital tem transformado a maneira como serviços e informações são intermediados, sendo as plataformas de eventos um caso notório.

No Contexto da Dispersão de Eventos e a Necessidade de Centralização, a proliferação de plataformas de mídia social e sites específicos contribuiu para a descentralização da informação sobre eventos. Embora a internet tenha ampliado o alcance da divulgação, ela gerou um excesso de fontes que, paradoxalmente, dificulta a curadoria e a descoberta. MELO e RIBEIRO (MELO, E. T.; RIBEIRO, J. F., 2020) afirmam que "a ausência de um agregador confiável e robusto impõe uma barreira de entrada informacional, limitando a participação do público em eventos menos mainstream". A solução para este desafio não se limita à agregação de dados, mas exige a padronização e o fornecimento de dados estruturados (local, data, tipo, classificação) que permitam filtros e buscas eficientes.

O trabalho propõe-se a preencher uma lacuna deixada por grandes plataformas globais, como Eventbrite e Sympla, que frequentemente negligenciam a granularidade e a especificidade de eventos de menor escala e de alcance estritamente local.

Pesquisas indicam que a eficácia da divulgação de eventos locais está diretamente ligada à sua capacidade de engajamento comunitário e à relevância contextual da informação. Um estudo de Silva e Oliveira (SILVA, A.; OLIVEIRA, B**.,** 2020) sobre a digitalização de eventos comunitários aponta que soluções focadas em geolocalização e filtros customizáveis aumentam significativamente a taxa de participação, superando a dispersão informacional comum em grupos de redes sociais e aplicativos de mensagens .

A tabela 1 a seguir compara o escopo das plataformas globais e o foco da solução proposta. O sucesso de uma plataforma local reside, portanto, na sua capacidade de ser um hub de informações confiável e acessível, integrando a gestão administrativa *(Web)* com a experiência do usuário final *(Mobile).*

Característica	Plataformas Globais (e.g., Eventbrite)	"Vamos Sair Hoje"
Escopo	Global/Nacional, Grandes Eventos	Estritamente Local, Eventos Comunitários
Foco	Venda de Ingressos, Logística de Grandes Eventos	Descoberta, Divulgação e Engajamento Comunitário
Tecnologia	Arquiteturas de Microsserviços, Cloud Computing	Arquitetura Híbrida (Web/Mobile) e Banco de Dados
Desafio	Escalabilidade de Transações	Consistência de Dados e UX Coesa

Tabela 1 – Comparação entre Plataformas Globais de Eventos e o Sistema "Vamos Sair Hoje".

4. Problema de Pesquisa

A ineficiência de soluções fragmentadas para divulgação de eventos locais, onde diferentes canais são utilizados sem integração, gerando inconsistência de dados, dificuldade de descoberta pelo público e baixo alcance para organizadores. O principal desafio consiste em desenvolver uma plataforma *web* juntamente com um dispositivo *mobile*, que ofereça uma experiência de usuário coesa, mantendo uma base de dados centralizada entre ambas plataformas, oferecendo tanto funcionalidades administrativas quanto de descoberta pública, sem comprometer o desempenho e a escalabilidade do sistema.

O problema de pesquisa pode ser sintetizado na seguinte pergunta:
Como desenvolver um sistema *web/mobile* eficiente, capaz de unificar as etapas de gerenciamento e divulgação de eventos locais, oferecendo uma experiência consistente e acessível através de navegadores web e dispositivos móveis?

5. Objetivo

Nesta seção são descritos os objetivos que fundamentam o desenvolvimento da solução proposta, abrangendo desde a definição das metas gerais do projeto até os desdobramentos específicos para cada uma de suas partes**.**

5.1 Objetivo Geral

Desenvolver, implementar e validar uma Plataforma *Web* de Gerenciamento de Eventos e um aplicativo *mobile*, unificados, utilizando tecnologias *web* consolidadas (*PHP, MySQL, Tailwind CSS*) e também tecnologias de dispositivos móveis (*Flutter*), com foco em prover uma solução escalável, de fácil manutenção e que otimize a experiência de organizadores e participantes.

O aplicativo foi implementado com *Flutter* e estruturado segundo uma Arquitetura Limpa, combinado com o padrão *BLoC (Business Logic Component)* para

o gerenciamento de estado (BLOC LIBRARY, 2025). O objetivo principal foi construir uma aplicação que permitisse ao usuário listar eventos, filtrar por tipo e cidade, visualizar detalhes e marcar eventos como favoritos, proporcionando uma experiência responsiva e modular que facilite manutenção e evolução. A implementação explora persistência local para favoritos e práticas de desacoplamento entre camadas. Resultados incluem um protótipo funcional, testes unitários, de *widgets* e de integração básicos e recomendações para trabalhos futuros que envolvam *backend*.

5.2 Objetivos Específicos

Organizamos os objetivos específicos deste trabalho em duas frentes principais do projeto: o sistema *web* administrativo e o aplicativo *mobile* voltado ao usuário final. Nos subconjuntos abaixo, descrevemos de forma clara, objetiva e focada nos resultados, as metas técnicas e funcionais necessárias para o desenvolvimento das duas plataformas que compõem a solução. As subseções a seguir detalham os objetivos relacionados a cada uma dessas partes.

5.2.1 Sistema Web

Esta subseção mostra os objetivos específicos relacionados ao desenvolvimento da plataforma web administrativa, responsável pelo gerenciamento de eventos, usuários e aos demais recursos do sistema. Os objetivos listados orientam as etapas de análise, estruturação, implementação e avaliação da solução web, garantindo que ela cumpra seu papel como núcleo de administração da plataforma.

- Identificar e definir os requisitos funcionais e não funcionais necessários ao funcionamento do sistema *web*.

- Estruturar o modelo de dados de forma a garantir integridade, consistência e eficiência no armazenamento das informações.

- Desenvolver uma interface responsiva e de fácil utilização, adequada a diferentes tipos de usuários.

- Estabelecer mecanismos de controle de acesso que assegurem níveis diferenciados de permissão.

- Adotar práticas de segurança e validação que minimizem vulnerabilidades e garantam a confiabilidade dos dados.

- Avaliar a experiência do usuário por meio de testes de usabilidade, propondo ajustes quando necessários.

- Verificar o desempenho e a escalabilidade do sistema em condições próximas ao ambiente real de operação.

5.2.2 Sistema Mobile

Nesta subseção apresentamos os objetivos específicos vinculados ao desenvolvimento do aplicativo *mobile*, voltado ao usuário final para a descoberta e interação com os eventos locais. Os objetivos aqui descritos estabelecem as diretrizes para a construção da interface, definição da arquitetura, implementação das funcionalidades principais e avaliação da experiência do usuário em um ambiente mobile.

- Construir um aplicativo mobile multiplataforma que permita acesso eficiente às funcionalidades principais do sistema.

- Definir uma arquitetura de software que favoreça organização, manutenibilidade e clareza entre interface, lógica e dados.

- Garantir que o design da interface seja responsivo e adequado aos diferentes tamanhos e orientações de tela.

- Implementar mecanismos de personalização, incluindo sistema de favoritos e persistência local de preferências.

- Assegurar práticas de proteção e validação dos dados manipulados pelo aplicativo.

- Desenvolver funcionalidades de busca e filtragem que ofereçam descoberta eficiente de eventos conforme critérios do usuário.

- Propor elementos de *feedback* visual que aprimorem a comunicação entre o sistema e o usuário durante interações.

- Realizar testes de usabilidade considerando características específicas da navegação mobile, como gestos e interações por toque.

6. Material e Métodos

A consecução do objetivo geral deste Trabalho de Conclusão de Curso — o desenvolvimento e validação de uma plataforma *web* e *mobile* para eventos locais — exigiu a adoção de uma metodologia de pesquisa e desenvolvimento rigorosa, capaz de conciliar a inovação tecnológica com a robustez acadêmica.

Esta seção detalha o tipo de pesquisa, a metodologia de desenvolvimento de *software* empregada, os materiais (tecnologias) utilizados e as etapas de validação do protótipo.

6.1. Tipo de Pesquisa e Abordagem Metodológica

O presente trabalho se enquadra como Pesquisa Aplicada, pois seu principal foco é gerar conhecimentos para aplicação prática, visando a solução de um problema específico: a dispersão e ineficiência na divulgação de eventos locais. A abordagem metodológica de desenvolvimento de *software* adotada foi a Metodologia Ágil, inspirada nos princípios do *Kanban* (TRELLO**,** 2025). A abordagem também é qualitativa e exploratória, visto que o sistema é construído e testado a partir de requisitos reais observados em usuários potenciais.

6.2. Arquitetura do Sistema e Materiais

O sistema "Vamos Sair Hoje" foi concebido sob uma Arquitetura Cliente-Servidor, onde o aplicativo móvel e a interface *web* pública atuam como clientes que consomem dados e serviços fornecidos por um servidor central. A comunicação entre todas as partes do sistema é intermediada diretamente com um banco de dados.

- **Ponto de Conexão entre o Backend Web e o Aplicativo Mobile**

A plataforma “Vamos Sair Hoje” adota uma arquitetura Cliente-Servidor na qual o painel *web* administrativo (PHP/MySQL) e o aplicativo *mobile* em *Flutter* compartilham o mesmo banco de dados *MySQL* hospedado remotamente. No entanto, a comunicação entre os sistemas não ocorre de maneira uniforme: enquanto a maior parte das operações do aplicativo é feita por meio de conexão direta ao banco utilizando a biblioteca *mysql1*, a autenticação — etapa crítica para segurança — é realizada exclusivamente através de ***endpoints REST* implementados em PHP**.

No *backend web*, a conexão com o banco de dados é gerenciada por uma classe centralizada baseada em *PDO (**PHP Data Objects**.)*, que controla acesso, tratamento de exceções e execução de operações administrativas (CRUD de eventos, anunciantes e locais). Já o aplicativo mobile utiliza uma classe de conexão *MySQL* configurada via variáveis de ambiente, permitindo executar consultas diretamente no banco e integrar seus resultados ao fluxo de estado usando o padrão *BLoC*.

Apesar desse acesso direto, as funcionalidades de **registro e *login*** do aplicativo não utilizam acesso ao banco pelo cliente. Em vez disso, foram implementadas duas rotas *REST* dedicadas responsáveis por validar credenciais, aplicar criptografia de senha e gerar *tokens JWT.*

Esses *endpoints* retornam respostas estruturadas em *JSON*, permitindo que o aplicativo gerencie a sessão do usuário de maneira segura, sem expor o banco de

dados nem manipular senhas diretamente através da camada *mobile*. Esse modelo híbrido reforça a segurança da autenticação, ao mesmo tempo em que mantém a simplicidade da arquitetura geral para as demais operações.

Essa abordagem, mesmo não seguindo integralmente o padrão de API centralizada, oferece um bom equilíbrio entre viabilidade técnica e segurança, permitindo o funcionamento integrado entre *web* e *mobile*. A arquitetura atual pode ser expandida futuramente para uma *API* completa, encapsulando todas as operações em *endpoints REST,* o que traria benefícios adicionais em segurança, padronização e escalabilidade.

- **Arquitetura de Sistemas *Web***

A arquitetura de *software web* é um dos pilares para o desenvolvimento de sistemas escaláveis e de fácil manutenção. No caso do "Vamos Sair Hoje", adotou-se uma abordagem tradicional e consolidada, utilizando PHP 8.x no *backend*, estruturado sob o padrão *MVC (Model-View-Controller)*, integrado com banco de dados *MySQL* e *frontend* responsivo com *Tailwind CSS*.

O padrão ***MVC (Model-View-Controller)***, amplamente descrito por Fowler (2002), propõe a separação das responsabilidades do sistema em três camadas principais: ***Model***, responsável pelos dados e pela lógica de negócio; ***View***, referente à interface com o usuário; e ***Controller***, responsável pelo controle e pelo fluxo da aplicação. Essa separação favorece a modularidade, além de facilitar a manutenção e a evolução do *software*.

- *Model*: gerencia dados e regras de negócio (PHP/MySQL)
- *View*: interface do usuário (HTML/CSS/JavaScript)
- *Controller*: lógica de controle e fluxo da aplicação (PHP)

A comunicação entre as camadas segue os princípios de Arquitetura Limpa, garantindo que o sistema tenha a possibilidade de ser expandido sem comprometer sua estabilidade. O uso de controle de versão com *Git* e práticas de desenvolvimento web modernas assegura o ciclo contínuo de entrega e melhoria do produto.

- **Arquitetura de Aplicações Móveis**

O sistema foi desenvolvido seguindo uma **arquitetura em camadas**, organizada em **apresentação**, **lógica de negócio** e **persistência**.

A camada de **apresentação** é composta por *widgets Flutter* organizados em páginas e componentes reutilizáveis. O **gerenciamento de estado** é realizado por

meio de ***BLoCs*** específicos (como *EventBloc*, *CityBloc* e *EventTypeBloc*), que expõem fluxos de eventos e estados para a interface do usuário (*UI*).

A camada de **domínio e serviços** contém as **interfaces de repositórios** e **serviços**, responsáveis por encapsular regras de negócio e definir contratos de acesso a dados. Já a camada de **dados** implementa os **repositórios concretos**, que obtêm informações de fontes locais (como arquivos *JSON* embutidos nos *assets*) e realizam a persistência local de favoritos por meio do ***Shared Preferences****.*

O fluxo geral de comunicação entre as camadas pode ser representado da seguinte forma:
UI → BLoC → Service → Repository → Data Source.

A arquitetura adotada foi a ***Clean Architecture*** (Arquitetura Limpa), conforme os princípios descritos por **Martin (2018)**, em conjunto com o ***BLoC Pattern***, seguindo as melhores práticas recomendadas para o desenvolvimento com ***Flutter*** (BLOC LIBRARY, 2025; FLUTTER TEAM, 2025).

A Arquitetura Limpa foi escolhida por promover uma estrutura modular, de fácil manutenção e com clara separação de responsabilidades entre as camadas de domínio, dados e apresentação. Essa organização facilita a testabilidade, reduz o acoplamento entre componentes e permite que cada parte do sistema evolua de forma independente.

O ***BLoC Pattern (Business Logic Component)***, criado pela equipe do *Google* e recomendado pela documentação oficial do *Flutter*, complementa essa arquitetura ao gerenciar o estado da aplicação de maneira **reativa e previsível**, por meio de ***streams*** que separam a lógica de negócio da interface do usuário.

A comunicação entre as camadas segue os princípios de **inversão de dependência** e **injeção de dependências**, garantindo que o sistema possa ser expandido e testado sem comprometer sua estabilidade.

Além disso, foram adotadas **boas práticas de engenharia de *software***, como **controle de versão com *Git***, **testes automatizados** e **integração contínua**, assegurando um ciclo constante de melhoria e entrega contínua do produto. A compilação nativa do *Flutter* mantém a **compatibilidade entre as plataformas *iOS* e *Android***, garantindo desempenho otimizado.

Os benefícios da **Arquitetura em Camadas** são diversos: o código torna-se mais **organizado, limpo e legível**, há **maior reutilização de componentes** e **facilidade de manutenção**, uma vez que alterações em uma camada não impactam diretamente as demais. Essa estrutura também favorece a **escalabilidade**, permitindo que o sistema cresça sem comprometer sua integridade e coerência estrutural.

- **Componentes da Arquitetura:**

1. **Camada *Core* (Domínio):**
    a. **Entidades(*Models*)** (Event, City, EventType): Classes que modelam os dados do domínio.
    b. **Exceções**: Tratamento específico de erros por contexto
    c. **Contratos(*Contracts*)**: Contém as interfaces dos repositórios (*repositories*) e serviços (*services*)

2. **Camada *Data* :**
    a. ***Repositories***: Implementam as interfaces contidas na camada de domínio e abstraem o acesso aos dados (APIs, banco local)
    b. ***Services***: Implementam as interfaces contidas na camada de domínio e fazem a comunicação com fontes externas e lógica de cache.
    c. ***Database*(db):** Faz a comunicação com o banco de dados e contém as queries necessárias para buscar os dados.

3. **Camada *UI* (Apresentação)**
    a. ***Views*** (Pages/Components): Interface construída com *widgets Flutter* que exibem dados de forma declarativa e capturam interações do usuário
    b. ***BLoCs***: Gerenciam estado e lógica de apresentação através de (BLOC LIBRARY, 2025):
    c. ***Events***: Representam ações do usuário *(LoadEvents, SearchEvents)*
    d. ***States***: Definem estados da UI *(Loading, Success, Failure)*
    e. ***Business Logic***: Processam eventos e emitem estados correspondentes

4. **Camada *Infrastructure* (Infra)**
    a. ***Providers*:** Provedores centralizado em um único arquivo para melhor entendimento e manutenibilidade
    b. ***Constants***: Variáveis do sistema que não se alteram.
    c. **Rotas (*Routes*)**: Rotas das páginas centralizadas.

6.2.1. Materiais (Tecnologias) Utilizados

O desenvolvimento do projeto empregou uma *stack* de tecnologias maduras e modernas, selecionadas com base em sua eficiência, robustez e capacidade de suportar o desenvolvimento multiplataforma.

A Tabela 2 apresenta o conjunto de tecnologias utilizadas no desenvolvimento do sistema, destacando suas versões, propósitos e justificativas técnicas. Essa organização permite visualizar de forma clara como cada componente contribui para a arquitetura geral, garantindo desempenho, segurança, compatibilidade e facilidade

de manutenção em todos os ambientes — local, servidor, *web, mobile, backend* e banco de dados.

Componente	Tecnologia	Versão	Justificativa Técnica
Ambiente Local	XAMPP	8.2.12	Utilização do XAMPP como ambiente de desenvolvimento local por integrar Apache, PHP e MySQL em um único pacote, facilitando testes, depuração e execução offline da aplicação antes da implantação no servidor, garantindo agilidade e compatibilidade com o ambiente de produção.
Servidor	HostGator com suporte a PHP e MySQL	-	Utilização da HostGator por oferecer ambiente LAMP (Linux, Apache, MySQL, PHP) compatível com o sistema, garantindo desempenho, estabilidade, segurança e suporte completo às tecnologias utilizadas (PHP e MySQL), além de painel cPanel e alta disponibilidade (99%).
Backend	PHP 8.x com arquitetura MVC	8x	Linguagem madura, alto desempenho em versões recentes, vasta documentação e facilidade de hospedagem para a camada de controle e lógica de negócio.
Banco de Dados	MySQL	8.0	Sistema de gerenciamento de banco de dados relacional (SGBDR) robusto, de código aberto, ideal para a integridade e persistência dos dados de eventos e usuários.
Web	HTML, CSS, JavaScript, Tailwind CSS	Últimas versões	Utilização de padrões web e o framework Tailwind CSS para garantir uma interface responsiva, moderna e de fácil manutenção no painel administrativo e na interface público.
Mobile	Flutter e Dart	3.7.2	Framework Google para desenvolvimento multiplataforma, que permite criar aplicativos nativos tanto para Android quanto para iOS a partir de uma única base de código.

Tabela 2 – Tecnologias, versões e justificativas utilizadas no desenvolvimento do sistema.

6.2.2. Considerações sobre Segurança de Dados

A segurança da informação constitui um eixo fundamental para a confiabilidade e robustez da plataforma “Vamos Sair Hoje”, especialmente considerando que o sistema realiza autenticação, armazenamento e manipulação de dados sensíveis, tanto no ambiente *web* quanto no aplicativo *mobile*. Em conformidade com boas práticas de Engenharia de *Software* e requisitos de aplicações, diversas medidas técnicas foram aplicadas para proteger a integridade, a confidencialidade e a disponibilidade dos dados.

a) Segurança no *Backend Web (PHP/MySQL)*

O *backend* foi implementado com foco em suavizar vulnerabilidades associadas a sistemas *web*. Para isso, foram adotadas as seguintes práticas:

- **Uso de *PDO* (*PHP Data Objects* - biblioteca nativa do *PHP*) com *Prepared Statements*** em 100% das operações *SQL*, impedindo ataques de injeção SQL e fazendo com que a consulta *SQL* seja compilada antes da inclusão dos dados.
- **Validação e sanitização de entradas** antes do processamento, reduzindo riscos de *XSS* (*Cross-Site Scripting*) e manipulação de parâmetros.

- **Controle de Acesso Baseado em Papéis (RBAC)**, garantindo que cada usuário (administrador, anunciante ou usuário comum) tenha acesso apenas às operações autorizadas.
- **Auditoria de ações críticas**, registrando modificações, logins, operações administrativas e interações sensíveis, aumentando a rastreabilidade.
- **Isolamento entre camadas (MVC)**, reduzindo a probabilidade de exposição acidental de funções internas.

O servidor de hospedagem utiliza um ambiente *LAMP (Linux, Apache, MySQL e PHP)*, garantindo compatibilidade e estabilidade para a aplicação. O acesso aos arquivos é realizado por SFTP, assegurando transferência criptografada e permissões controladas. A infraestrutura da HostGator aplica camadas adicionais de proteção, reforçando a segurança operacional do sistema.

b) Segurança no Aplicativo *Mobile (Flutter)*

A camada mobile adota mecanismos para proteger o fluxo de informações e evitar que dados sensíveis sejam expostos na aplicação cliente. Entre as práticas adotadas, destacam-se:

- **Autenticação exclusivamente por meio de *endpoints REST* dedicados**, impedindo que o dispositivo *mobile* manipule senhas diretamente no banco de dados.
- **Uso de *JWT (JSON Web Token)*** com verificação de validade, promovendo autenticação segura e padronizada.
- **Tratamento de erros, exceções e respostas *HTTP***, garantindo que falhas não exponham informações internas da aplicação.
- **Arquitetura em camadas (Arquitetura Limpa)**, isolando a lógica de negócio e reduzindo o risco de acesso indevido a serviços internos.

Apesar de algumas consultas de leitura serem feitas diretamente no banco, nenhuma operação sensível (*login*, cadastro ou validação de permissões) ocorre sem passar por *endpoints* autenticados, reduzindo a superfície de ataque no ambiente mobile.

c) Segurança do Banco de Dados

O banco *MySQL* foi estruturado fora do diretório público do servidor, garantindo que arquivos de configuração e credenciais não possam ser acessados externamente. Entre as medidas aplicadas:

- **Usuários de banco de dados com permissões mínimas necessárias**, evitando privilégios excessivos.
- **Configuração de *collation* segura e validação de tipos**, reduzindo riscos de armazenamento incorreto ou corrompido.

- **Integridade referencial com chaves estrangeiras**, evitando registros órfãos e inconsistência de dados.
- **Autenticação obrigatória e restrições de IP** no ambiente remoto.

d) Segurança no fluxo de Autenticação e Autorização

A autenticação foi projetada conforme princípios utilizados em APIs modernas. O fluxo implementado segue:

1. O usuário envia credenciais via *HTTPS* para o endpoint de *login*.
2. O *backend* valida a senha criptografada e gera um ***token JWT* assinado**.
3. O *token* é armazenado apenas em memória no aplicativo mobile.
4. *Endpoints* protegidos validam o *token* antes de permitir qualquer operação.
5. A expiração automática exige renovação do *token*, reduzindo riscos de *session hijacking.*

Esse processo impede o acesso direto ao banco e adiciona uma camada de abstração e segurança entre cliente e servidor.

e) Medidas de Proteção no Ambiente de Hospedagem

Para reforçar a segurança em nível de infraestrutura, foram adotadas boas práticas oferecidas pelo servidor *HostGator*, incluindo:

- **HTTPS obrigatório** com certificados ativos.
- **Bloqueio de portas externas**, exceto as necessárias ao funcionamento da aplicação.
- **SFTP para *deploy***, evitando protocolos inseguros como FTP padrão.
- **Permissões de arquivo restritas**, protegendo o *backend* contra leitura pública.

f) Governança, Riscos e Recomendações Futuras

Apesar das medidas implementadas, algumas evoluções são recomendadas para ampliar a maturidade de segurança da solução:

- Migração gradual para ***API REST* centralizada**, eliminando o acesso direto do mobile ao banco de dados.
- Implementação de ***rate limiting*** para reduzir tentativas de *login* automatizadas.
- Inserção de ***logs* centralizados** e monitoramento contínuo de erros.
- Uso de **criptografia *HTTPS* mútua *(SSL/TLS)*** entre serviços internos.
- Adoção de ***CORS* configurado**, garantindo que apenas origens autorizadas acessem o *backend*.

A combinação destas práticas garante integridade operacional, diminui riscos de ataques e posiciona a plataforma de forma segura e escalável para evoluções futuras.

6.3. Etapas de Desenvolvimento e Validação

O desenvolvimento do protótipo funcional foi executado em ciclos iterativos, com foco na entrega contínua das funcionalidades principais.

6.3.1. Modelagem e Implementação do *Backend*

A etapa de modelagem de dados resultou na definição das tabelas principais (Eventos, Usuários, Locais, Favoritos) e seus respectivos relacionamentos, garantindo a integridade referencial no *MySQL*. A implementação do *backend* em *PHP* seguiu os princípios do *MVC*, onde os *Controllers* foram responsáveis por receber as requisições da *API*, e os Models por interagir com o banco de dados de forma segura, utilizando *Prepared Statements* para mitigar ataques de injeção *SQL*.

6.3.2 – Uso do Trello para Gestão do Desenvolvimento do Projeto

Para organizar as nossas tarefas e ter um acompanhamento do progresso do desenvolvimento, utilizamos a ferramenta *Trello*. Optamos por utilizá-lo por ser um sistema online de gestão de projetos que é baseado em quadros visuais. Nele podemos criar listas e cartões para representar atividades, o que facilita o controle das etapas do desenvolvimento e a priorização das entregas conforme as devidas prioridades (TRELLO, 2025).

Durante todo o desenvolvimento do sistema *“Vamos Sair Hoje”*, o Trello foi utilizado para registrar, acompanhar e monitorar as atividades do projeto. As tarefas foram organizadas em listas que representavam os diferentes estágios do fluxo de trabalho, como *“A Fazer”*, *“Em Progresso”* e *“Concluído”*. Por ter uma abordagem mais visual, possibilitou ter uma visão clara do andamento do projeto, contribuindo para uma melhor distribuição das atividades e otimização do tempo.

Além disso, o uso do *Trello* proporcionou maior controle, organização e transparência no processo de desenvolvimento, permitindo o acompanhamento contínuo da evolução do sistema e o registro histórico das entregas. A ferramenta mostrou-se eficiente para apoiar a metodologia incremental adotada, auxiliando no cumprimento dos prazos e na gestão das demandas de cada etapa.

6.3.3. Desenvolvimento do Web

O desenvolvimento envolveu a criação de um banco de dados relacional robusto com entidades como eventos, locais, anunciantes, tipos de evento e usuários. Foi implementado um sistema completo de CRUD (Criação, Leitura, Atualização e Exclusão) para gerenciar todas as entidades do sistema, incluindo *upload* de imagens e vídeos para divulgação dos eventos. A plataforma conta com sistema de

autenticação de usuários (comum e estabelecimentos), permitindo cadastro, login e gerenciamento de perfis. Implementou-se um sistema de filtragem dinâmica de eventos por cidade, data e tipo, com carregamento assíncrono via *AJAX* para melhor experiência do usuário.

A interface foi desenvolvida com design responsivo utilizando *Tailwind CSS*, garantindo usabilidade em dispositivos móveis e *desktop*. Adicionalmente, foi criado um sistema de favoritos, auditoria de eventos, carrossel de propaganda para anunciantes e integração com *Google Analytics.* O controle de versão foi realizado através do *Git/GitHub* com *branches* específicas para cada funcionalidade implementada. O projeto utiliza *PDO* para conexão segura com banco de dados *MySQL*, prevenindo *SQL Injection*, e implementa tratamento de erros e validações tanto no *frontend* quanto no *backend*.

6.3.4. Desenvolvimento do Frontend Mobile (Flutter)

O desenvolvimento *mobile* priorizou uma **arquitetura limpa**, combinada com o **padrão *BLoC/Cubit***, garantindo a reatividade da interface e a manutenção estruturada do código. As principais etapas contemplaram a implementação das camadas de **Dados**, **Domínio** e **Apresentação**, prática amplamente utilizada que assegura **organização**, **escalabilidade** e **facilidade de manutenção**.

6.3.4.1 Camada de Domínio *(Domain Layer)*

A implementação iniciou-se pela **Camada de Domínio**, responsável por centralizar as regras de negócio da aplicação e também definir os *models* (entidades de domínio), que representam os objetos centrais do sistema de forma independente de *frameworks* ou tecnologias externas. Nessa etapa foram definidas as **entidades de domínio**,que representam a lógica central do problema sem dependências de tecnologias externas. Essas entidades contêm apenas os atributos e regras essenciais, garantindo baixo acoplamento e maior coesão.

Também foram estabelecidos os **contratos de repositório**, garantindo que a lógica de negócio permaneça independente da forma como os dados seriam persistidos ou obtidos. Essa abordagem aumenta a flexibilidade, estabilidade e capacidade de evolução do sistema.

6.3.4.2 Camada de Dados *(Data Layer)*

Após a definição do domínio, foi desenvolvida a **Camada de Dados**, responsável pela implementação concreta dos repositórios e serviços definidos na camada anterior. Nessa etapa foram criadas as classes responsáveis pela comunicação com bancos de dados, *APIs* ou fontes locais, além dos modelos de dados utilizados internamente na persistência. Também foram incorporadas rotinas de

tratamento de erros e exceções, garantindo integridade e consistência durante o tráfego das informações.

6.3.4.3 Camada de Apresentação *(Presentation Layer)*

Por fim, foi implementada a **Camada de Apresentação**, responsável pela interface visual e interação com o usuário. A organização da *UI* seguiu princípios de usabilidade, compondo *widgets* e componentes reutilizáveis. O gerenciamento de estado adotou o padrão ***BLoC***, que integra a lógica de negócio da camada de domínio à interface de forma reativa e previsível (MARTIN, 2019; BLOC LIBRARY, 2025; FLUTTER TEAM, 2025).

Durante todo o processo, foi aplicada uma **metodologia incremental e iterativa**, com ciclos curtos de desenvolvimento, em que cada incremento entregava **funcionalidades testáveis**. O **versionamento de código** foi realizado com ***Git***, assegurando controle e rastreabilidade das versões.

O ambiente de desenvolvimento incluiu o ***Flutter SDK*** (compatível com ***Dart***), o ***Android Studio*** e o ***Visual Studio Code*** para edição de código, além de **emuladores e dispositivos reais** para testes manuais.

As principais etapas do desenvolvimento foram: **levantamento de requisitos**, **modelagem de dados**, **implementação da camada de dados e repositórios**, **criação dos serviços**, **construção da interface de usuário (*UI*)**, **integração via *BLoC*** e **testes de integração**.

6.3.4.4 Usabilidade (*UX*) e *Design System* Adaptativo (*Dark/Light Mode*)

No desenvolvimento do ***frontend mobile***, foram aplicadas práticas de **Usabilidade (*UX*)** com o objetivo de proporcionar uma **experiência fluida, intuitiva e agradável** ao usuário. A interface foi projetada com base em princípios de **hierarquia visual, contraste adequado, legibilidade e acessibilidade**, assegurando que o usuário consiga realizar suas ações com o mínimo de esforço cognitivo.

Além disso, o sistema foi estruturado com um ***Design System*** **adaptativo**, capaz de alternar automaticamente entre os modos **claro (*Light*)** e **escuro (*Dark*)**, conforme as **preferências do usuário** ou as **configurações do dispositivo**. Essa abordagem promove **conforto visual** em diferentes condições de iluminação e mantém a **consistência estética** em toda a aplicação.

A definição de **cores, espaçamentos, tipografia e componentes reutilizáveis** seguiu um padrão centralizado, implementado por meio de **themes personalizados no *Flutter***. Essa padronização garante **uniformidade visual** entre as telas e facilita a **manutenção e evolução** do projeto.

O uso de um ***Design System* adaptativo** também contribui para a **escalabilidade** da aplicação, permitindo a introdução de novos componentes visuais sem comprometer o padrão de identidade definido. Dessa forma, o aplicativo não apenas cumpre suas funções técnicas, mas também se destaca pela **qualidade da experiência do usuário (*UX*)** e pelo **cuidado com o design responsivo**.

A figura 1 ilustra a diferença entre os **temas claro e escuro** implementados na interface.

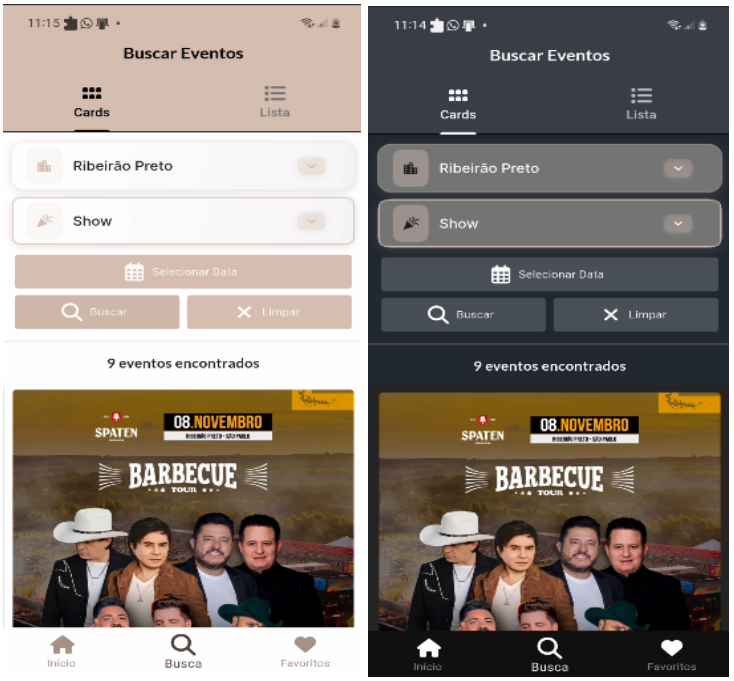


Figura 1 – Exemplo de adaptação entre os modos Light e Dark do aplicativo.

A figura 1 apresenta a tela de busca de eventos do aplicativo nas versões modo claro e modo escuro. A principal diferença entre elas está na paleta de cores utilizada: enquanto o modo claro prioriza tons suaves e fundos claros para melhor visualização em ambientes iluminados, o modo escuro utiliza tons escuros e maior contraste, proporcionando maior conforto visual em locais com pouca luz e reduzindo o consumo de energia em dispositivos com telas *OLED*. Essa adaptação demonstra a aplicação do *Design System* adaptativo, que ajusta automaticamente a interface conforme as preferências do usuário ou o tema do sistema.

6.3.4. Uso do *GoRouter* para Navegação no *Flutter*

A navegação entre páginas é um dos aspectos mais importantes e essenciais em aplicações móveis, pois influencia diretamente a experiência do usuário e a organização das rotas dentro do sistema. Originalmente, o *Flutter* oferece um

mecanismo de navegação nativo baseado em *Navigator* e *MaterialPageRoute*, que apesar de ser muito importante e funcional, ele acaba tornando muito complexo em aplicações de médio e grande porte por causa do alto número de rotas aninhadas e à dificuldade de gerenciamento do histórico de navegação.

Com o objetivo de simplificar esse processo e adotar uma abordagem mais declarativa e escalável, optou-se pela utilização do *GoRouter*, biblioteca oficial que é mantida pela equipe do Flutter (FLUTTER TEAM, 2024). O *GoRouter* nos oferece uma *API* mais moderna e intuitiva, com suporte nativo a rotas nomeadas, navegação reativa, redirecionamentos condicionais, *Deep Linking* e integração com o estado da aplicação, o que o torna ideal para projetos estruturados com gerenciamento de estado via *BLoC*.

Ao contrário do que faz a navegação tradicional nativa do *Flutter*, o *GoRouter* nos dá a liberdade de definir todas as rotas de forma centralizada e declarativa, favorecendo a legibilidade e a manutenção do código. Além disso, ele suporta navegação baseada em *URL*, o que facilita a integração com a *Web*.

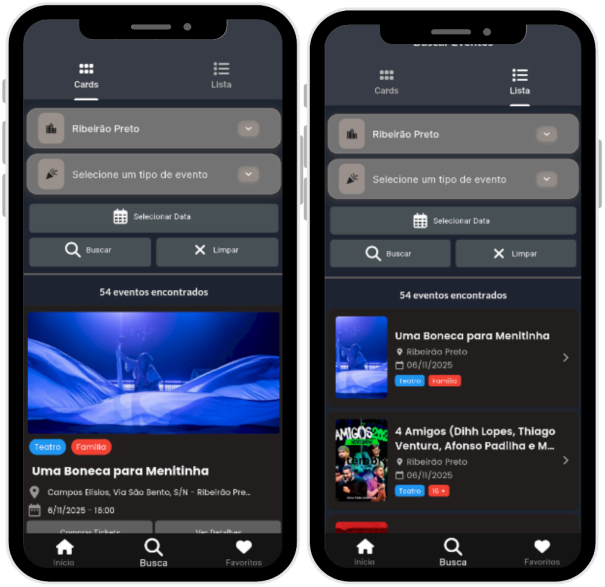
Essa escolha arquitetural proporciona:

- Maior organização e previsibilidade nas transições de tela;
- Suporte a rotas aninhadas e parâmetros dinâmicos de forma simplificada;
- Integração direta com o gerenciamento de estado, permitindo redirecionamentos automáticos (por exemplo, para tela de *login* quando o usuário não estiver autenticado);
- Alta escalabilidade, pois o controle de rotas permanece desacoplado dos *widgets*.

Página Inicial

A Figura 2 apresenta a página de entrada do aplicativo, disponível nos modos claro e escuro, cuja alternância pode ser realizada por meio do botão localizado acima do rodapé. Essa página inicial exibe a logo do sistema, as categorias de eventos, as imagens dos patrocinadores e os cards dos eventos disponíveis. Cada categoria permite o filtro de eventos correspondentes, redirecionando o usuário para uma página específica. Ao selecionar um card, são exibidos os detalhes completos do evento. Além disso, é possível visualizar todas as categorias e eventos disponíveis por meio das opções "Ver mais" e "Ver todos".

Figura 2 – Página Inicial do aplicativo com tema claro e escuro disponíveis.



Página de Pesquisa

A Figura 3 apresenta a página de pesquisa, na qual o usuário pode buscar eventos aplicando filtros por tipo de evento, cidade e data. Os resultados podem ser exibidos em formato de cards ou em lista, conforme a preferência do usuário. Assim como na página inicial, ao selecionar um evento, o usuário é redirecionado para uma página contendo todos os detalhes correspondentes.

Figura 3 – Página de Pesquisa disponibilizada para busca de eventos.

Autenticação

A Figura 4 apresenta a tela de login do aplicativo, na qual o usuário pode acessar sua conta inserindo e-mail e senha cadastrados. A interface foi desenvolvida com foco em simplicidade, trazendo apenas os elementos essenciais para facilitar o acesso rápido à plataforma. Além disso, há opções para recuperação de senha e para redirecionamento à página de registro, permitindo que novos usuários criem uma conta caso ainda não possuam cadastro

Figura 4 – Página de Login e Página de Cadastro.

6.3.5. Estratégia de Testes e Validação

A validação e testes do sistema mobile ocorreram em três níveis: Testes de Unidade, Testes de *Widgets* e Testes de Integração.

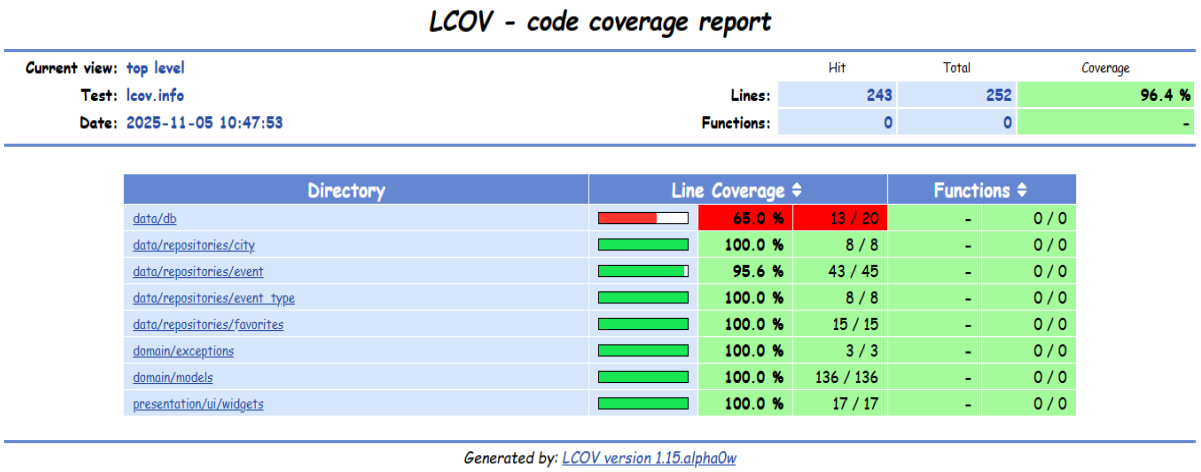
Para garantir a qualidade e a robustez do código-fonte, foram implementados testes de unidade e de *widgets* utilizando a biblioteca de testes nativa do *Flutter*. A eficácia dessa abordagem foi quantificada por meio da ferramenta *LCOV*. O *LCOV (Line Coverage)* é uma ferramenta do *Flutter* que faz a medição da quantidade de

linhas de código que estão cobertas por testes, mostrando quais partes do aplicativo foram testadas, ajudando o desenvolvedor a identificar quais partes do código ainda não possuem testes. Isso gerou um relatório detalhado da cobertura de testes e que foi transformado em um gráfico para melhor compreensão. O resultado demonstrou uma cobertura geral de 96,4% das linhas de código, indicando que a grande maioria da lógica de negócio, modelos de dados e componentes de interface foi rigorosamente verificada.

A análise do relatório mostra que diretórios críticos como *core/models, core/exceptions, data/repositories* e *ui/widgets* alcançaram 100% de cobertura, assegurando que as regras de negócio e a manipulação de dados estão funcionando como esperado. A única área com cobertura inferior (*core/db* com 65%) foi identificada como um ponto de melhoria para iterações futuras, embora não tenha comprometido a funcionalidade essencial do protótipo.

Além dos testes automatizados, foram realizados testes de integração entre as camadas da aplicação, bem como uma validação com usuários para simular o uso real. A avaliação de usabilidade revelou uma alta taxa de compreensão das interfaces e fluidez na navegação entre telas.

A combinação dessas abordagens metodológicas, aliada à validação quantitativa da cobertura de testes, permitiu a entrega de um protótipo funcional que não apenas atende aos objetivos propostos, mas também serve como uma base sólida e confiável para trabalhos futuros.



LCOV - code coverage report

Current view: top level | Test: lcov.info | Date: 2025-11-05 10:47:53

	Hit	Total	Coverage
Lines:	243	252	96.4 %
Functions:	0	0	-

Directory	Line Coverage		Functions	
data/db	65.0 %	13 / 20	-	0 / 0
data/repositories/city	100.0 %	8 / 8	-	0 / 0
data/repositories/event	95.6 %	43 / 45	-	0 / 0
data/repositories/event_type	100.0 %	8 / 8	-	0 / 0
data/repositories/favorites	100.0 %	15 / 15	-	0 / 0
domain/exceptions	100.0 %	3 / 3	-	0 / 0
domain/models	100.0 %	136 / 136	-	0 / 0
presentation/ui/widgets	100.0 %	17 / 17	-	0 / 0

Generated by: LCOV version 1.15.alpha0w

Figura 5 – Relatório de cobertura de testes gerado pela ferramenta LCOV, demonstrando 96,4% de cobertura de código

A figura 5 apresenta o relatório de cobertura de testes (*code coverage*) gerado pelo *LCOV* no ambiente *Flutter*. O relatório mostra que 96,4% do código-fonte foi coberto por testes automatizados, demonstrando um alto nível de confiabilidade e manutenção do sistema. Cada diretório listado exibe a porcentagem de linhas

testadas, permitindo identificar módulos com menor cobertura — como core/db, com 65% — e outros com cobertura total. Esse tipo de análise é fundamental para avaliar a qualidade do código e garantir a estabilidade da aplicação, reduzindo a ocorrência de erros em produção.

6.3.5 Testes de Usabilidade

Para complementar os testes técnicos e assegurar que a plataforma ofereça uma experiência satisfatória aos usuários, foram realizados testes de usabilidade contemplando tanto a interface *web* quanto o aplicativo *mobile*. O objetivo foi avaliar a facilidade de uso, a clareza das informações apresentadas e a eficiência com que as tarefas principais podiam ser executadas.

Os testes foram conduzidos com o proprietário do sistema, responsável pela contratação e definição das funcionalidades. A participação dele foi fundamental, pois representa o perfil do usuário administrador que utilizará o sistema diariamente para gerenciar os eventos. Durante o processo, o avaliador interagiu livremente com as interfaces, realizando tarefas como cadastro e edição de eventos, navegação entre telas, visualização de informações e uso de filtros, enquanto eram observados eventuais dificuldades, dúvidas e comportamentos inesperados.

Para orientar a análise, foram considerados critérios como:

- Intuitividade da navegação: facilidade para localizar menus e funcionalidades.
- Consistência visual: coerência entre telas, elementos, ícones e padrões definidos no *Design System*.
- Eficiência: tempo e passos necessários para concluir uma tarefa.
- *Feedback* ao usuário: clareza das mensagens, alertas e estados de carregamento.

Ao término dos testes, o proprietário aprovou o funcionamento geral do sistema, destacando a boa fluidez das telas e a organização dos elementos visuais. Também foram levantadas sugestões de melhorias — principalmente relacionadas ao ajuste de textos, posicionamento de botões e otimização de fluxos específicos — que foram incorporadas ao projeto antes da finalização.

Os testes de usabilidade permitiram validar não apenas o funcionamento das interfaces, mas também a adequação da experiência do usuário às expectativas do público-alvo e às necessidades do administrador, reforçando a qualidade da solução desenvolvida.

6.3.6 Integração Contínua (*Continuous Integration*) com *GitHub Actions*

Durante o desenvolvimento do sistema, foi implementado um fluxo de **i**ntegração contínua utilizando o ***GitHub Actions***, com o objetivo de automatizar

tarefas recorrentes do ciclo de desenvolvimento e garantir a qualidade do código. Isso está presente tanto no sistema *web* quanto no sistema *mobile*.

A integração contínua é um processo que automatiza e executa testes, validações e compilações a cada novo *commit*, *pull request* ou *push* permitindo detectar erros precocemente e manter a estabilidade do projeto. Essa prática reduz falhas em produção, melhora a colaboração entre desenvolvedores e assegura entregas mais rápidas e consistentes.

No caso deste projeto, as *pipelines* configuradas no **GitHub Actions** executavam as seguintes etapas principais:

- **Pipeline do Sistema Web:**

A *pipeline* denominada *Deploy to HostGator* tem como objetivo automatizar o processo de implantação do projeto diretamente no servidor da *HostGator*, garantindo agilidade e padronização no fluxo de publicação.

Sua execução é configurada para ocorrer automaticamente sempre que houver um *push* na *branch* principal (*main*) do repositório, utilizando o ambiente de execução *Ubuntu* disponibilizado pelo *GitHub Actions*.

A primeira etapa do processo, denominada *Checkout repository*, utiliza a ação *actions/checkout@v3*, responsável por clonar o código-fonte do repositório na máquina virtual de execução, tornando-o disponível para as etapas seguintes.

Em seguida, a etapa *Deploy via SFTP* realiza a transferência dos arquivos para o servidor de hospedagem, por meio da ação *appleboy/scp-action@v0.1.7*. Essa ação estabelece uma conexão segura via protocolo SFTP, utilizando a porta XXXX, e autentica-se com o usuário e a senha armazenada de forma segura nos *GitHub Secrets* (SFTP_PASSWORD).

Durante essa etapa, todos os arquivos do diretório raiz do projeto (./*) são enviados para o caminho remoto correspondente ao diretório público do servidor *HostGator*. Além disso, a opção *use_insecure_cipher: true* é ativada para assegurar a compatibilidade com servidores que utilizam algoritmos de criptografia mais antigos, evitando falhas de conexão durante o processo de *deploy*.

- **Pipeline do Sistema *Mobile*:**

A *pipeline* denominada *Dart* tem como objetivo automatizar o processo de integração contínua do projeto *Flutter*, garantindo a validação do código-fonte por meio de análises estáticas e testes automatizados.

Sua execução é configurada para ser acionada automaticamente em dois cenários: sempre que ocorre um *push* ou um *pull request* direcionado à *branch*

principal (*main*). O ambiente de execução utilizado é o *Ubuntu*, disponibilizado pelo *GitHub Actions*, o que assegura uma configuração padronizada e independente do sistema operacional local dos desenvolvedores.

Durante a execução, são definidas variáveis de ambiente que simulam as credenciais e parâmetros necessários para a conexão com o banco de dados *MySQL*, além de uma *URL* base para *uploads*. Essas variáveis são gravadas em um arquivo *.env* gerado dinamicamente, possibilitando que o ambiente de *build* reflita as mesmas condições da aplicação em produção ou em testes.

A primeira etapa utiliza a ação *actions/checkout@v4* para clonar o repositório na máquina virtual, garantindo o acesso ao código-fonte. Em seguida, a ação *subosito/flutter-action@v2* é responsável por configurar o ambiente do *Flutter* na versão 3.29.3, dentro do canal estável (*stable*), assegurando consistência entre os ambientes de desenvolvimento e integração.

Na sequência, a etapa gerar arquivo .env cria o arquivo de variáveis de ambiente a partir dos valores definidos anteriormente, permitindo que o projeto seja executado e testado com as configurações adequadas.

Posteriormente, a etapa Instalar dependências executa o comando *flutter pub get*, responsável por baixar e configurar todas as dependências do projeto. Logo após, a etapa analisar código realiza uma verificação estática com o comando *flutter analyze*, identificando possíveis erros de sintaxe, padrões incorretos e boas práticas de código.

Por fim, a etapa Rodar testes executa o comando *flutter test*, o qual realiza os testes automatizados definidos no projeto, garantindo a integridade das funcionalidades implementadas e evitando regressões.

Essa automação proporciona maior confiabilidade ao processo de desenvolvimento, reduz o risco de erros em produção e assegura a qualidade contínua do código antes da etapa de implantação.

Essa automação possibilitou maior controle sobre as versões e garantiu que apenas código validado fosse integrado ao repositório principal, aumentando a confiabilidade do sistema.

A escolha pelo ***GitHub Actions*** se deu por ser uma ferramenta nativa da plataforma *GitHub*, oferecendo integração direta com o repositório, configuração simples por meio de arquivos *YAML* e ampla compatibilidade com linguagens e *frameworks* modernos, como o *Flutter*

7. Análise Comparativa do Sistema Proposto com Plataformas de Mercado

7.1 Visão Geral, Escopo e Comparativo Técnico

As plataformas dominantes no mercado, como *Eventbrite e Sympla*, possuem um escopo vasto e escalável, focando em eventos de grande porte com complexidade transacional e alta emissão de bilhetes. Tecnicamente, elas utilizam *tech stacks* complexas, incluindo *Python (Django), React*, e uma infraestrutura robusta na *Amazon Web Services (AWS)* com *data warehouses* para análise de dados massivos. Em contraste, o sistema “Vamos Sair Hoje” diferencia-se por seu escopo vertical e estritamente local, focado na unificação, organização e descoberta de eventos de menor escala, resolvendo o problema da dispersão de informações (MELO; RIBEIRO, 2020). O projeto opta pela eficiência e consistência de código: o *back-end* em *PHP/MySQL* oferece estabilidade e baixo custo operacional, enquanto o *Flutter* no *front-end* garante consistência de interface de usuário (*UX*) e agilidade de manutenção via única base de código (MOURA; NUNES, 2023; SILVA, 2023).

7.2 Comparativo Funcional e Profundidade da Gestão

A distinção mais significativa do sistema desenvolvido está na sua capacidade de oferecer um Ecossistema de Gestão Unificada, que vai além da simples bilheteria. Enquanto as plataformas de mercado delegam o valor à transação e relatórios financeiros, o sistema proposto prioriza o Controle Completo do Ciclo de Vida do Evento pelo organizador, por meio de um Sistema de Gerenciamento de Conteúdo (CMS) completo. O *back-office* desenvolvido em *PHP/MySQL* oferece CRUD completo não apenas para Eventos, mas também para Patrocinadores e Locais de Eventos, demonstrando maior granularidade de dados.

Ademais, o projeto atende a rigorosos requisitos acadêmicos de Sistemas de Informação e Segurança: a implementação de um sistema de Autenticação Multi-nível (Administrador, CNPJ, Usuário) e módulos de Auditoria de conteúdo são diferenciais que atestam a rastreabilidade e a segurança dos dados internos (FREITAS, 2023). Nenhum sistema de mercado com foco em transação delega tal nível de controle e granularidade a entidades como patrocinadores, sendo o principal valor técnico-funcional desta pesquisa.

7.3 Diferenciais do Sistema e Conclusão

Os principais diferenciais do sistema Vamos Sair Hoje são sua arquitetura MVVM/BLoC, que garante a separação limpa da lógica de negócio e performance (LIMA; COSTA, 2022), o *Design Adaptativo* com *Dark/Light Mode* (melhorando a acessibilidade e a usabilidade), e a otimização das *queries* para a descoberta de eventos locais (ROCHA; CARVALHO, 2023).

Em conclusão, o projeto posiciona-se de forma estratégica ao resolver um problema de nicho — a fragmentação da informação local — com rigor técnico. Ele se apresenta não como um concorrente das plataformas transacionais, mas sim como uma solução vertical de gestão e descoberta que aplica conceitos avançados de Engenharia de *Software* para criar um *framework* robusto de *back-office* (Web) e uma

experiência de consumo fluida (*Mobile*). A viabilidade do projeto é atestada pela aplicação bem-sucedida da *stack* tecnológica enxuta e eficiente para resolver problemas práticos de governança e usabilidade (GOMES; FERREIRA, 2023).

8. Considerações Finais

8.1 Limitações do Estudo e Sugestões para Trabalhos Futuros

Apesar da conclusão bem-sucedida do desenvolvimento de ambas plataformas, o presente trabalho enfrentou uma limitação técnica relevante inerente ao ecossistema de desenvolvimento da *Apple*. Devido à ausência de *hardware* compatível (máquina com sistema operacional *macOS*) para a execução do processo de build e assinatura do aplicativo móvel, não foi possível gerar o artefato final (.ipa) para testes de validação na plataforma *iOS*, conforme exigido pelas políticas de distribuição da *Apple*. Essa restrição limitou a validação prática da usabilidade e performance em um dos sistemas operacionais móveis almejados.

Como sugestão para trabalhos futuros, recomenda-se a validação completa do aplicativo móvel em ambiente *iOS* e a publicação nas respectivas lojas de aplicativos, garantindo assim a abrangência total da solução *cross-platform*.

9 – Resultados e Discussão

O resultado do projeto foi uma solução multiplataforma completa capaz de permitir que usuários descubram eventos locais através de uma interface pública *web* e *mobile*, enquanto organizadores e administradores gerenciam conteúdo através de um painel administrativo robusto. O sistema oferece recursos de categorização por tipo de evento, sistema de autenticação com múltiplos níveis de acesso e funcionalidades completas de CRUD.

O aplicativo *mobile*, desenvolvido em *Flutter* utilizando arquitetura limpa complementa a plataforma oferecendo aos usuários uma experiência nativa otimizada para dispositivos móveis. A aplicação implementa gerenciamento de estado com *BLoC (Business Logic Component)*, garantindo separação clara entre lógica de negócio e interface, facilitando manutenção e testes. O app conta com sistema de favoritos utilizando *SharedPreferences* para persistência local, banco de dados *SQLite* para *cache* de eventos e sincronização *offline*, e navegação intuitiva com acesso rápido às funcionalidades principais como busca de eventos por cidade, filtros por tipo de evento, visualização de detalhes e gerenciamento de favoritos.

Durante os testes, observou-se que tanto a aplicação *web* quanto o aplicativo *mobile* apresentaram desempenho satisfatório. A plataforma *web* mostrou tempos de carregamento adequados e interface responsiva funcionando corretamente em diferentes dispositivos e navegadores, enquanto o app mobile demonstrou fluidez na

navegação, carregamento eficiente de imagens através de *cache*, e experiência *offline* satisfatória com dados previamente sincronizados. A implantação no servidor *HostGator* confirmou a compatibilidade do sistema com ambientes de hospedagem compartilhada, mantendo performance estável mesmo com recursos limitados. O sistema de gerenciamento administrativo mostrou-se intuitivo para usuários técnicos e não-técnicos.

A análise comparativa com soluções existentes revelou que o "Vamos Sair Hoje" se diferencia por seu foco específico em eventos locais e informais, interface simplificada disponível em múltiplas plataformas (*web e mobile*), sistema de auditoria que garante qualidade do conteúdo publicado, e arquitetura escalável que permite evolução futura do sistema com inclusão de novas funcionalidades tanto no *backend* quanto nas aplicações cliente.

As demais seções permanecem inalteradas, pois as menções ao *HostGator* são específicas apenas para o ambiente de produção e testes, não afetando a fundamentação teórica ou metodológica do trabalho.

Referências

MELO, E. T.; RIBEIRO, J. F. Sistemas de Gestão de Eventos: Desafios na Dispersão de Informações e Soluções de Centralização. Journal of Digital Communication, v. 4, n. 1, p. 22-40, 2020.

PARKER, G. G.; VAN ALSTYNE, M. W.; CHOUDARY, S. P. Platform Revolution: How Networked Markets Are Transforming the Economy and How to Make Them Work for You. W. W. Norton & Company, 2016.

BECK, K. et al. Manifesto para Desenvolvimento Ágil de Software. 2001. Disponível em: https://agilemanifesto.org/. Acesso em: 20 out. 2025.

SILVA, A.; OLIVEIRA, B. Digitalização de Eventos Comunitários: Um Estudo de Caso. Revista de Sistemas de Informação, v. 12, n. 3, 2020.

GOOGLE. *Arquitetura de Aplicativos Flutter* [*Flutter App Architecture*]. Disponível em: https://docs.flutter.dev/app-architecture. Acesso em: 4 nov. 2025.

BLOC LIBRARY. *Guia de Início — BLoC Library*. Disponível em: https://bloclibrary.dev/pt-br/getting-started/. Acesso em: 28 out. 2025.

FOWLER, M. *Padrões de Arquitetura de Aplicações Corporativas* [*Patterns of Enterprise Application Architecture*]. Boston: Addison-Wesley, 2002.

MARTIN, R. C. *Arquitetura Limpa: o guia do artesão para estrutura e design de software* [*Clean Architecture: A Craftsman's Guide to Software Structure and Design*]. São Paulo: Alta Books, 2019.

NIELSEN, Jakob. *Engenharia de Usabilidade.* Boston: Academic Press, 1994.

GARRETT, Jesse James. *Os Elementos da Experiência do Usuário: design centrado no usuário para a web e além.* 2. ed. Berkeley: New Riders, 2011.

APPLE INC. *Diretrizes de Interface Humana: Modo Escuro.* Cupertino: Documentação Apple Developer, 2025. Disponível em: https://developer.apple.com/design/human-interface-guidelines. Acesso em: 4 nov. 2025.

GOOGLE MATERIAL DESIGN. *Material Design 3: Design e Temas Adaptativos.* Mountain View: Google, 2025. Disponível em: https://m3.material.io. Acesso em: 4 nov. 2025.

T**RELLO.** *Guia do Trello: Trabalhe de forma mais colaborativa e produtiva com quadros digitais.* Atlassian, 2025. Disponível em: https://trello.com. Acesso em: 4 nov. 2025.

EQUIPE FLUTTER. *GoRouter: Roteamento declarativo para aplicações Flutter.* Google Developers, 2024. Disponível em: https://pub.dev/packages/go_router.

EQUIPE FLUTTER. *Navegação e roteamento no Flutter.* Google Developers, 2025.

FREITAS, S. O. Protocolos de Autenticação e Autorização em Aplicações Distribuídas. São Paulo: Editora Atlas, 2023.