Instituto Federal de Rondônia - Campus Ji-Paraná

Analise e Desenvolvimento de Sistemas

MURAL COMUNITÁRIO

Desenvolvimento de Aplicativo Mobile para Comunicação Comunitária Utilizando React Native e Google Sheets

Trabalho Desenvolvimento para Dispositivos Móveis

Autor: Daniel Cristian Amorim Rocha

Docente: Karan Luciano Silva

Ji-Paraná - RO
18/07/2025

Conteúdo

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho apresenta o desenvolvimento de um aplicativo mobile denominado "Mural Comunitário", uma solução tecnológica inovadora para facilitar a comunicação e o compartilhamento de informações dentro de comunidades locais. O projeto utiliza React Native como framework principal para desenvolvimento mobile multiplataforma, integrando-se com Google Sheets como backend simplificado, eliminando a necessidade de infraestrutura complexa de servidor.

A metodologia empregada baseou-se em princípios de desenvolvimento ágil, com foco na simplicidade, acessibilidade e baixo custo de implementação. O aplicativo permite que comunidades visualizem, busquem e filtrem eventos, avisos e informações relevantes de forma intuitiva e eficiente. A arquitetura proposta demonstra a viabilidade de soluções tecnológicas simples para problemas reais de comunicação comunitária.

Os resultados obtidos evidenciam que a integração entre React Native e Google Sheets oferece uma alternativa viável e econômica para organizações comunitárias com recursos limitados. O sistema desenvolvido apresenta interface responsiva, funcionalidades de busca avançada, sistema de categorização e tratamento adequado de erros, proporcionando uma experiência de usuário satisfatória.

As contribuições deste trabalho incluem a demonstração prática de como tecnologias acessíveis podem ser aplicadas para resolver problemas sociais, a criação de uma arquitetura replicável para projetos similares, e a validação de Google Sheets como backend viável para aplicações de pequeno e médio porte.

Palavras-chave: React Native, Google Sheets, Aplicativo Mobile, Comunicação Comunitária, Desenvolvimento de Software.

Abstract

This work presents the development of a mobile application called "Community Board", an innovative technological solution to facilitate communication and information sharing within local communities. The project uses React Native as the main framework for cross-platform mobile development, integrating with Google Sheets as a simplified backend, eliminating the need for complex server infrastructure.

The methodology employed was based on agile development principles, focusing on simplicity, accessibility and low implementation cost. The application allows communities to view, search and filter events, notices and relevant information in an intuitive and efficient way. The proposed architecture demonstrates the viability of simple technological solutions for real community communication problems.

The results obtained show that the integration between React Native and Google Sheets offers a viable and economical alternative for community organizations with limited resources. The developed system features responsive interface, advanced search functionality, categorization system and adequate error handling, providing a satisfactory user experience.

The contributions of this work include the practical demonstration of how accessible technologies can be applied to solve social problems, the creation of a replicable architecture for similar projects, and the validation of Google Sheets as a viable backend for small and medium-sized applications.

Keywords: React Native, Google Sheets, Mobile Application, Community Communication, Software Development.

1 Introdução

A comunicação efetiva dentro de comunidades locais representa um desafio constante na sociedade contemporânea. Com o crescimento urbano acelerado e a diversificação dos meios de comunicação, muitas comunidades enfrentam dificuldades para manter seus membros informados sobre eventos, avisos importantes e atividades locais relevantes. Tradicionalmente, essas informações eram disseminadas através de murais físicos, cartazes ou comunicação boca a boca, métodos que apresentam limitações significativas em termos de alcance, atualização e acessibilidade.

O advento das tecnologias móveis e da internet criou novas oportunidades para revolucionar a forma como as comunidades se comunicam internamente. Smartphones tornaram-se ubíquos, com penetração superior a 80% da população brasileira segundo dados da Pesquisa Nacional por Amostra de Domicílios Contínua (PNAD Contínua) de 2023 [1]. Esta realidade tecnológica oferece um terreno fértil para o desenvolvimento de soluções digitais que possam atender às necessidades específicas de comunicação comunitária.

No entanto, muitas organizações comunitárias operam com recursos financeiros e técnicos limitados, o que torna inviável a implementação de soluções tecnológicas complexas e custosas. Sistemas tradicionais de desenvolvimento de aplicativos frequentemente requerem infraestrutura de servidor dedicada, equipes técnicas especializadas e investimentos substanciais em manutenção e hospedagem. Estas barreiras econômicas e técnicas impedem que muitas comunidades se beneficiem das vantagens oferecidas pela tecnologia digital.

Neste contexto, surge a necessidade de explorar abordagens alternativas que combinem eficácia tecnológica com simplicidade de implementação e baixo custo operacional. O presente trabalho propõe uma solução inovadora que utiliza Google Sheets como backend para um aplicativo mobile desenvolvido em React Native, criando uma arquitetura que elimina a necessidade de infraestrutura complexa de servidor enquanto mantém funcionalidades robustas de comunicação comunitária.

1.1 Justificativa

A escolha por desenvolver uma solução tecnológica para comunicação comunitária justifica-se por múltiplos fatores convergentes. Primeiramente, observa-se uma lacuna significativa entre as necessidades de comunicação das comunidades e as soluções tecnológicas disponíveis no mercado. Aplicativos comerciais existentes frequentemente são projetados para audiências amplas e genéricas, não atendendo às especificidades e necessidades particulares de comunidades locais.

Além disso, a pandemia de COVID-19 acelerou a digitalização de processos comunitários, evidenciando a importância de canais de comunicação digital eficientes. Comu-

nidades que anteriormente dependiam exclusivamente de encontros presenciais e comunicação física foram forçadas a adaptar-se rapidamente a meios digitais, revelando tanto oportunidades quanto desafios nesta transição.

Do ponto de vista técnico, a evolução das tecnologias de desenvolvimento mobile, particularmente frameworks como React Native, democratizou o acesso ao desenvolvimento de aplicativos multiplataforma. Simultaneamente, a popularização de serviços em nuvem gratuitos, como Google Sheets, criou oportunidades para arquiteturas inovadoras que reduzem drasticamente os custos de implementação e manutenção.

A relevância acadêmica deste trabalho reside na exploração de arquiteturas não convencionais para desenvolvimento de software, demonstrando como a criatividade na combinação de tecnologias existentes pode resultar em soluções eficazes e economicamente viáveis. Esta abordagem contribui para o corpo de conhecimento em engenharia de software, especificamente nas áreas de arquitetura de sistemas e desenvolvimento mobile.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver um aplicativo mobile multiplataforma para comunicação comunitária que utilize React Native como framework de desenvolvimento e Google Sheets como backend, demonstrando a viabilidade de soluções tecnológicas simples e de baixo custo para problemas reais de comunicação em comunidades locais.

1.2.2 Objetivos Específicos

O presente trabalho estabelece os seguintes objetivos específicos, organizados de forma a abordar tanto aspectos técnicos quanto sociais da solução proposta:

1. **Projetar e implementar uma arquitetura de sistema** que integre React Native com Google Sheets, estabelecendo um padrão replicável para projetos similares que necessitem de backend simplificado e de baixo custo.

2. **Desenvolver uma interface de usuário intuitiva e responsiva** que atenda aos princípios de usabilidade e acessibilidade, garantindo que usuários com diferentes níveis de familiaridade tecnológica possam utilizar o aplicativo efetivamente.

3. **Implementar funcionalidades avançadas de busca e filtragem** que permitam aos usuários localizar rapidamente informações relevantes dentro do mural comunitário, incluindo busca textual e filtros por categoria.

4. **Criar um sistema de categorização flexível** que permita a organização lógica de diferentes tipos de conteúdo comunitário, facilitando a navegação e a localização de informações específicas.

5. **Estabelecer mecanismos robustos de tratamento de erros** e feedback ao usuário, garantindo uma experiência consistente mesmo em condições adversas de conectividade ou falhas de sistema.

6. **Validar a eficácia da solução proposta** através de testes funcionais e de usabilidade, demonstrando que a arquitetura escolhida atende adequadamente aos requisitos de comunicação comunitária.

7. **Documentar detalhadamente o processo de desenvolvimento** e as decisões arquiteturais tomadas, criando um guia replicável para futuras implementações similares.

1.3 Estrutura do Trabalho

Este documento está organizado em capítulos que abordam progressivamente todos os aspectos do desenvolvimento do aplicativo Mural Comunitário, desde a fundamentação teórica até a implementação prática e análise de resultados.

O Capítulo 2 apresenta a fundamentação teórica necessária para compreensão do trabalho, abordando conceitos relacionados ao desenvolvimento mobile com React Native, integração com APIs web, princípios de design de interface e arquiteturas de software para aplicações móveis. Esta seção também inclui uma revisão da literatura relevante sobre comunicação comunitária digital e soluções tecnológicas para organizações com recursos limitados.

O Capítulo 3 detalha a metodologia empregada no desenvolvimento do projeto, incluindo a escolha das tecnologias, o processo de design da arquitetura, as decisões de implementação e os critérios de avaliação estabelecidos. Esta seção também aborda as considerações éticas e de privacidade relevantes ao projeto.

O Capítulo 4 apresenta o desenvolvimento prático do sistema, descrevendo detalhadamente a implementação de cada componente, as soluções técnicas adotadas para desafios específicos e as iterações realizadas durante o processo de desenvolvimento. Inclui também diagramas de arquitetura, fluxogramas de funcionamento e exemplos de código relevantes.

O Capítulo 5 analisa os resultados obtidos, apresentando testes de funcionalidade, métricas de performance, avaliações de usabilidade e validação da arquitetura proposta. Esta seção também discute as limitações identificadas e compara a solução desenvolvida com alternativas existentes no mercado.

Finalmente, o Capítulo 6 apresenta as conclusões do trabalho, sintetizando as contribuições realizadas, discutindo as implicações dos resultados obtidos e propondo direções para trabalhos futuros. Esta seção também reflete sobre o potencial de replicação da solução em outros contextos comunitários.

2 Fundamentação Teórica

2.1 Desenvolvimento Mobile Multiplataforma

O desenvolvimento de aplicações móveis multiplataforma emergiu como uma resposta às demandas crescentes por soluções que funcionem consistentemente em diferentes sistemas operacionais móveis. Tradicionalmente, o desenvolvimento mobile requeria a criação de aplicações nativas separadas para cada plataforma, utilizando linguagens e ferramentas específicas como Swift para iOS e Java/Kotlin para Android [3].

Esta abordagem tradicional, embora oferecesse performance otimizada e acesso completo às funcionalidades nativas dos dispositivos, apresentava desvantagens significativas em termos de custo de desenvolvimento, tempo de implementação e manutenção de código. Organizações precisavam manter equipes especializadas para cada plataforma, duplicar esforços de desenvolvimento e sincronizar atualizações entre diferentes bases de código.

O paradigma de desenvolvimento multiplataforma surgiu para endereçar essas limitações, propondo frameworks que permitem a criação de aplicações que funcionam em múltiplas plataformas a partir de uma única base de código. Esta abordagem oferece vantagens substanciais em termos de eficiência de desenvolvimento, redução de custos e consistência de experiência do usuário entre plataformas.

2.1.1 React Native: Arquitetura e Princípios

React Native, desenvolvido pelo Facebook e lançado em 2015, representa uma evolução significativa no desenvolvimento mobile multiplataforma [4]. Diferentemente de abordagens baseadas em WebView, que essencialmente executam aplicações web dentro de um container nativo, React Native utiliza uma arquitetura híbrida que combina JavaScript com componentes nativos reais.

A arquitetura fundamental do React Native baseia-se em três camadas principais: a camada JavaScript, onde reside a lógica de negócio e interface do usuário; a camada de bridge, responsável pela comunicação assíncrona entre JavaScript e código nativo; e a camada nativa, que executa componentes de interface e funcionalidades específicas da plataforma.

Esta arquitetura permite que desenvolvedores escrevam a maior parte da aplicação em JavaScript, utilizando conceitos familiares do desenvolvimento web, enquanto mantêm acesso a funcionalidades nativas e performance próxima à de aplicações nativas tradicionais. O framework utiliza o paradigma de programação declarativa, onde desenvolvedores descrevem como a interface deve aparecer em diferentes estados, e o sistema gerencia automaticamente as atualizações necessárias.

O sistema de componentes do React Native baseia-se no conceito de Virtual DOM,

adaptado para o contexto mobile. Quando o estado da aplicação muda, React Native calcula as diferenças necessárias e atualiza apenas os componentes afetados, otimizando performance e responsividade da interface.

2.1.2 Vantagens e Limitações do Desenvolvimento Multiplataforma

O desenvolvimento multiplataforma oferece vantagens significativas que justificam sua adoção crescente na indústria de software mobile. A principal vantagem reside na eficiência de desenvolvimento, permitindo que uma única equipe desenvolva aplicações para múltiplas plataformas simultaneamente. Esta eficiência traduz-se em redução substancial de custos de desenvolvimento, especialmente relevante para organizações com recursos limitados.

A consistência de experiência do usuário representa outra vantagem importante. Aplicações multiplataforma naturalmente mantêm consistência visual e funcional entre plataformas, reduzindo a curva de aprendizado para usuários que utilizam diferentes dispositivos. Esta consistência também simplifica processos de design e teste, reduzindo a complexidade geral do projeto.

A velocidade de desenvolvimento e time-to-market constituem fatores críticos em mercados competitivos. Frameworks multiplataforma permitem iterações mais rápidas e deployment simultâneo em múltiplas plataformas, acelerando significativamente o processo de lançamento de produtos.

No entanto, o desenvolvimento multiplataforma também apresenta limitações que devem ser consideradas. Performance pode ser inferior à de aplicações nativas, especialmente para aplicações que requerem processamento intensivo ou animações complexas. O acesso a funcionalidades específicas da plataforma pode ser limitado ou requerer desenvolvimento de módulos nativos adicionais.

A dependência de frameworks terceirizados introduz riscos relacionados à evolução e suporte contínuo dessas tecnologias. Mudanças nas plataformas nativas podem requerer atualizações nos frameworks multiplataforma, potencialmente causando incompatibilidades temporárias.

2.2 Integração com APIs Web e Serviços em Nuvem

A integração com APIs web representa um aspecto fundamental do desenvolvimento de aplicações móveis modernas. APIs (Application Programming Interfaces) permitem que aplicações se comuniquem com serviços externos, acessem dados remotos e integrem funcionalidades de terceiros [5].

No contexto de aplicações móveis, a integração com APIs web é essencial para funcionalidades como sincronização de dados, autenticação de usuários, processamento de pagamentos e acesso a serviços especializados. Esta integração permite que aplicações

móveis transcendam as limitações de armazenamento e processamento local, aproveitando recursos e dados disponíveis na nuvem.

2.2.1 Protocolos de Comunicação e Formatos de Dados

A comunicação entre aplicações móveis e APIs web baseia-se predominantemente no protocolo HTTP/HTTPS, utilizando métodos padronizados como GET, POST, PUT e DELETE para diferentes tipos de operações. O protocolo HTTPS é especialmente importante para garantir segurança na transmissão de dados sensíveis.

JSON (JavaScript Object Notation) emergiu como o formato de dados predominante para comunicação entre aplicações web e móveis devido à sua simplicidade, legibilidade e suporte nativo em JavaScript [6]. Comparado a formatos alternativos como XML, JSON oferece menor overhead de dados e parsing mais eficiente, características particularmente importantes em contextos móveis onde largura de banda e processamento podem ser limitados.

A estrutura de dados JSON permite representação hierárquica de informações de forma intuitiva, facilitando tanto a serialização quanto a deserialização de objetos complexos. Esta característica é especialmente relevante para aplicações que manipulam dados estruturados como eventos, usuários e configurações.

2.2.2 Google Sheets como Backend Alternativo

Google Sheets representa uma abordagem não convencional mas viável para backend de aplicações, especialmente adequada para projetos com requisitos específicos de simplicidade e baixo custo [7]. Esta plataforma oferece funcionalidades de banco de dados básicas através de uma interface familiar de planilha, eliminando a necessidade de configuração e manutenção de infraestrutura de servidor tradicional.

A API do Google Sheets permite acesso programático aos dados armazenados, suportando operações de leitura e escrita através de requisições HTTP padrão. Para casos de uso que requerem apenas leitura de dados, Google Sheets oferece a funcionalidade de exportação CSV pública, que pode ser acessada diretamente sem autenticação, simplificando significativamente a arquitetura da aplicação.

Esta abordagem apresenta vantagens substanciais para projetos comunitários e educacionais. A interface familiar de planilha permite que usuários não técnicos gerenciem dados diretamente, reduzindo a dependência de desenvolvedores para tarefas administrativas básicas. O modelo de custo do Google Sheets é extremamente favorável para aplicações de pequeno e médio porte, com limites generosos para uso gratuito.

As limitações desta abordagem incluem restrições de performance para grandes volumes de dados, funcionalidades limitadas de consulta e relacionamento entre dados, e dependência da infraestrutura do Google. No entanto, para muitos casos de uso, es-

pecialmente aqueles envolvendo dados relativamente simples e volumes moderados, estas limitações são aceitáveis considerando os benefícios oferecidos.

2.3 Design de Interface para Aplicações Móveis

O design de interface para aplicações móveis requer considerações específicas que diferem significativamente do design para web ou desktop. As limitações de tamanho de tela, métodos de interação baseados em toque e contextos de uso variados criam desafios únicos que devem ser endereçados através de princípios de design especializados.

2.3.1 Princípios de Usabilidade Mobile

A usabilidade em contextos móveis é influenciada por fatores únicos como mobilidade do usuário, limitações de atenção, variabilidade de condições ambientais e diversidade de dispositivos. Jakob Nielsen identificou princípios fundamentais de usabilidade que, quando adaptados para contextos móveis, fornecem diretrizes essenciais para design efetivo [8].

O princípio de simplicidade torna-se ainda mais crítico em interfaces móveis devido às limitações de espaço e atenção. Interfaces devem priorizar funcionalidades essenciais, minimizar elementos desnecessários e apresentar informações de forma hierárquica clara. A regra de "três cliques"sugere que usuários devem conseguir acessar qualquer funcionalidade principal em no máximo três interações.

Feedback imediato é essencial para manter usuários engajados e informados sobre o estado do sistema. Em contextos móveis, onde conexões podem ser instáveis, indicadores de carregamento, confirmações de ações e mensagens de erro claras são particularmente importantes.

A consistência visual e funcional reduz a curva de aprendizado e aumenta a eficiência de uso. Padrões estabelecidos de interface, como navegação por abas, gestos padrão e convenções visuais, devem ser respeitados para aproveitar o conhecimento prévio dos usuários.

2.3.2 Material Design e Sistemas de Design

Material Design, desenvolvido pelo Google, representa um sistema de design abrangente que estabelece princípios, componentes e diretrizes para criação de interfaces digitais consistentes e intuitivas [9]. Este sistema baseia-se em metáforas físicas, utilizando conceitos como profundidade, movimento e luz para criar interfaces que se comportam de forma previsível e natural.

Os princípios fundamentais do Material Design incluem o uso de superfícies e elevação para criar hierarquia visual, animações significativas que guiam a atenção do usuário,

e um sistema de cores e tipografia que promove legibilidade e acessibilidade. Estes princípios são especialmente relevantes para aplicações móveis, onde a clareza visual é essencial devido às limitações de tamanho de tela.

O sistema de componentes do Material Design oferece elementos de interface pré-definidos e testados, como botões, cards, barras de navegação e campos de entrada. Estes componentes seguem padrões estabelecidos de comportamento e aparência, reduzindo o trabalho de design e desenvolvimento enquanto garantem consistência com expectativas dos usuários.

A implementação do Material Design em React Native é facilitada por bibliotecas como React Native Paper, que fornecem componentes pré-construídos que seguem as especificações do Material Design. Esta abordagem permite que desenvolvedores criem interfaces profissionais sem necessidade de design customizado extensivo.

2.4 Arquiteturas de Software para Aplicações Móveis

A arquitetura de software para aplicações móveis deve considerar características específicas deste ambiente, incluindo limitações de recursos, conectividade intermitente, variedade de dispositivos e necessidades de sincronização de dados. Diferentes padrões arquiteturais foram desenvolvidos para endereçar esses desafios específicos.

2.4.1 Arquitetura Cliente-Servidor

A arquitetura cliente-servidor representa o padrão mais comum para aplicações móveis que requerem acesso a dados remotos. Nesta arquitetura, a aplicação móvel funciona como cliente, fazendo requisições a um servidor remoto que processa as solicitações e retorna dados ou confirmações.

Esta arquitetura oferece vantagens significativas em termos de centralização de dados, sincronização entre dispositivos, backup automático e capacidade de processamento. O servidor pode implementar lógica de negócio complexa, validações de dados e integrações com sistemas externos, liberando o cliente móvel para focar na apresentação e interação com o usuário.

No entanto, a dependência de conectividade de rede representa uma limitação importante desta arquitetura. Aplicações puramente cliente-servidor podem tornar-se inutilizáveis em situações de conectividade limitada ou ausente. Esta limitação levou ao desenvolvimento de arquiteturas híbridas que combinam processamento local com sincronização remota.

2.4.2 Padrões de Sincronização de Dados

A sincronização de dados entre aplicações móveis e servidores remotos requer estratégias específicas para lidar com conectividade intermitente, conflitos de dados e oti-

mização de performance. Diferentes padrões foram desenvolvidos para endereçar esses desafios.

O padrão de sincronização pull-based envolve a aplicação móvel solicitando atualizações do servidor em intervalos regulares ou em resposta a ações do usuário. Este padrão é simples de implementar e oferece controle direto sobre quando e quais dados são sincronizados. No entanto, pode resultar em uso desnecessário de largura de banda e bateria se não for implementado cuidadosamente.

Estratégias de cache local permitem que aplicações funcionem offline, armazenando dados localmente e sincronizando com o servidor quando conectividade está disponível. Esta abordagem melhora significativamente a experiência do usuário em condições de conectividade variável, mas introduz complexidade adicional relacionada à resolução de conflitos e consistência de dados.

2.5 Comunicação Comunitária Digital

A digitalização da comunicação comunitária representa uma evolução natural dos métodos tradicionais de disseminação de informações em grupos locais. Pesquisas em comunicação comunitária identificam características específicas que diferenciam este contexto de comunicação corporativa ou de massa [10].

2.5.1 Características da Comunicação Comunitária

A comunicação comunitária caracteriza-se por sua natureza local, participativa e orientada para ação coletiva. Diferentemente da comunicação de massa, que segue um modelo unidirecional de transmissão, a comunicação comunitária enfatiza diálogo, feedback e construção colaborativa de significado.

A confiança representa um elemento fundamental na comunicação comunitária. Membros da comunidade tendem a valorizar informações provenientes de fontes conhecidas e confiáveis dentro da própria comunidade. Esta característica influencia significativamente o design de sistemas de comunicação comunitária, que devem incorporar mecanismos para estabelecer e manter credibilidade das fontes de informação.

A relevância local é outro aspecto distintivo. Informações compartilhadas em contextos comunitários devem ter aplicabilidade direta para a vida dos membros da comunidade. Sistemas de comunicação comunitária devem, portanto, incorporar mecanismos de filtragem e categorização que permitam aos usuários identificar rapidamente informações relevantes para suas necessidades específicas.

2.5.2 Tecnologia e Inclusão Digital

A implementação de soluções tecnológicas para comunicação comunitária deve considerar questões de inclusão digital e acessibilidade. Comunidades frequentemente apre-

sentam diversidade significativa em termos de familiaridade tecnológica, recursos econômicos e acesso a dispositivos modernos.

Design inclusivo torna-se essencial para garantir que soluções tecnológicas não excluam membros da comunidade com menor familiaridade tecnológica. Interfaces devem ser intuitivas, utilizar linguagem clara e fornecer feedback adequado para guiar usuários através de diferentes funcionalidades.

A consideração de limitações de conectividade é particularmente importante em contextos comunitários, onde acesso à internet de alta velocidade pode ser limitado ou intermitente. Aplicações devem ser otimizadas para funcionar adequadamente em condições de conectividade variável, minimizando uso de dados e fornecendo funcionalidade offline quando possível.

3 Metodologia

3.1 Abordagem de Desenvolvimento

O desenvolvimento do aplicativo Mural Comunitário seguiu uma metodologia híbrida que combina elementos de desenvolvimento ágil com práticas de prototipagem rápida, adaptada especificamente para o contexto acadêmico e as limitações de recursos disponíveis. Esta abordagem foi escolhida considerando a natureza exploratória do projeto, que visa validar uma arquitetura não convencional, e a necessidade de iterações rápidas para refinamento da solução.

A metodologia empregada baseia-se nos princípios do Manifesto Ágil, priorizando software funcionando sobre documentação abrangente, colaboração com o cliente sobre negociação de contratos, resposta a mudanças sobre seguir um plano, e indivíduos e interações sobre processos e ferramentas [2]. Estes princípios foram adaptados para o contexto de desenvolvimento individual acadêmico, onde o "cliente"representa a comunidade-alvo e os requisitos emergem através de análise de necessidades e validação conceitual.

3.1.1 Fases de Desenvolvimento

O processo de desenvolvimento foi estruturado em cinco fases distintas, cada uma com objetivos específicos e critérios de conclusão bem definidos. Esta estruturação permitiu progresso sistemático enquanto mantinha flexibilidade para ajustes baseados em descobertas e aprendizados durante o desenvolvimento.

A **Fase de Análise e Planejamento** concentrou-se na definição de requisitos, análise de viabilidade técnica e design inicial da arquitetura. Esta fase incluiu pesquisa de tecnologias disponíveis, análise de soluções existentes no mercado, e definição de critérios de sucesso para o projeto. O resultado principal desta fase foi um documento de especificação técnica que serviu como guia para as fases subsequentes.

A **Fase de Prototipagem** envolveu a criação de protótipos funcionais para validar conceitos técnicos e de usabilidade. Protótipos foram desenvolvidos para testar a integração com Google Sheets, validar a arquitetura de comunicação proposta, e explorar diferentes abordagens de design de interface. Esta fase foi crucial para identificar desafios técnicos antecipadamente e refinar a abordagem de implementação.

A **Fase de Desenvolvimento Core** concentrou-se na implementação das funcionalidades principais do aplicativo, incluindo visualização de dados, sistema de busca e filtragem, e interface de usuário básica. Esta fase seguiu práticas de desenvolvimento iterativo, com ciclos curtos de implementação, teste e refinamento.

A **Fase de Refinamento e Otimização** focou na melhoria da experiência do usuário, otimização de performance, implementação de tratamento de erros robusto, e polimento da interface. Esta fase também incluiu testes extensivos em diferentes disposi-

tivos e condições de rede.

A **Fase de Validação e Documentação** envolveu testes finais de funcionalidade, validação da arquitetura proposta, e criação de documentação técnica e acadêmica abrangente. Esta fase também incluiu análise crítica dos resultados obtidos e identificação de limitações e oportunidades de melhoria.

3.2 Seleção de Tecnologias

A seleção de tecnologias para o projeto foi guiada por critérios específicos que refletem tanto os objetivos técnicos quanto as limitações práticas do contexto acadêmico. Os critérios principais incluíram facilidade de aprendizado e implementação, disponibilidade de documentação e suporte da comunidade, custo de implementação e manutenção, e alinhamento com os objetivos de demonstrar viabilidade de soluções simples e acessíveis.

3.2.1 Framework de Desenvolvimento Mobile

A escolha do React Native como framework principal foi baseada em múltiplos fatores convergentes. Primeiramente, React Native oferece desenvolvimento multiplataforma genuíno, permitindo que uma única base de código funcione tanto em iOS quanto Android, maximizando o alcance da solução desenvolvida. Esta característica é particularmente importante para aplicações comunitárias, onde usuários podem utilizar dispositivos de diferentes fabricantes e sistemas operacionais.

A curva de aprendizado relativamente suave do React Native, especialmente para desenvolvedores com experiência em JavaScript e React, foi outro fator decisivo. A sintaxe familiar e os conceitos de programação declarativa facilitaram o desenvolvimento rápido e a iteração eficiente durante o processo de criação do aplicativo.

A disponibilidade de bibliotecas e componentes pré-construídos no ecossistema React Native acelerou significativamente o desenvolvimento. Bibliotecas como React Native Paper forneceram componentes de interface que seguem padrões estabelecidos de design, reduzindo o tempo necessário para criar uma interface profissional e consistente.

A performance do React Native, embora não equivalente à de aplicações nativas puras, mostrou-se adequada para os requisitos do projeto. Para uma aplicação focada em exibição de informações e interações básicas de usuário, a performance oferecida pelo React Native atende completamente às necessidades identificadas.

3.2.2 Solução de Backend

A decisão de utilizar Google Sheets como backend representa uma escolha arquitetural não convencional que requer justificação detalhada. Esta escolha foi motivada principalmente pelo objetivo de demonstrar viabilidade de soluções de baixo custo e simplicidade de implementação para organizações comunitárias.

Google Sheets oferece funcionalidades de banco de dados básicas através de uma interface familiar que não requer conhecimento técnico especializado. Esta característica permite que administradores comunitários gerenciem conteúdo diretamente, sem dependência de desenvolvedores ou sistemas complexos de gerenciamento de conteúdo.

O modelo de custo do Google Sheets é extremamente favorável para aplicações de pequeno e médio porte. Os limites para uso gratuito são generosos o suficiente para atender a maioria das comunidades locais, eliminando custos recorrentes de hospedagem e manutenção de servidor.

A API pública de exportação CSV do Google Sheets simplifica significativamente a arquitetura da aplicação, eliminando necessidades de autenticação, gerenciamento de sessões, e configuração de servidor. Esta simplicidade reduz pontos de falha e facilita manutenção e troubleshooting.

As limitações desta abordagem foram cuidadosamente consideradas e julgadas aceitáveis para o escopo do projeto. Limitações de performance para grandes volumes de dados, funcionalidades limitadas de consulta, e dependência da infraestrutura do Google são trade-offs aceitáveis considerando os benefícios oferecidos.

3.3 Design da Arquitetura

A arquitetura do sistema foi projetada seguindo princípios de simplicidade, modularidade e manutenibilidade. O design arquitetural reflete o objetivo de criar uma solução que seja tanto tecnicamente robusta quanto conceitualmente simples, demonstrando que arquiteturas não convencionais podem ser eficazes para casos de uso específicos.

3.3.1 Arquitetura Geral do Sistema

O sistema segue uma arquitetura cliente-servidor simplificada, onde o aplicativo React Native funciona como cliente e Google Sheets fornece dados através de sua API pública de exportação CSV. Esta arquitetura elimina a necessidade de um servidor intermediário tradicional, reduzindo complexidade e custos operacionais.

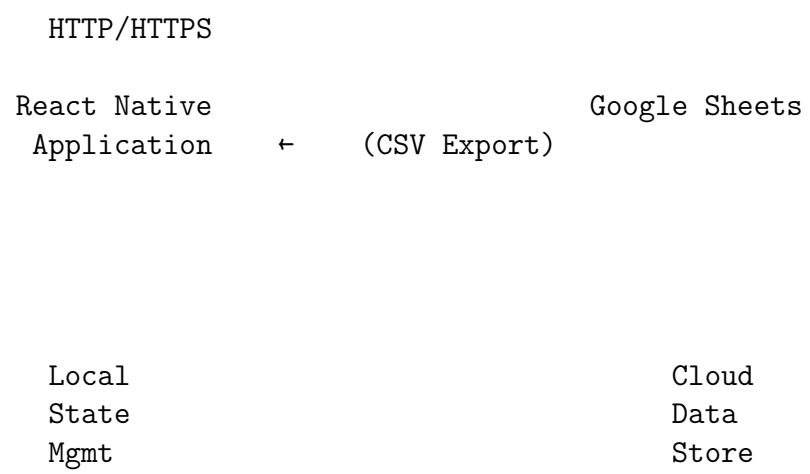

Figura 1: Arquitetura Geral do Sistema

A comunicação entre cliente e servidor utiliza protocolo HTTP/HTTPS padrão, com o cliente fazendo requisições GET para o endpoint de exportação CSV do Google Sheets. Os dados retornados em formato CSV são processados localmente pelo cliente usando a biblioteca Papa Parse, que converte CSV em objetos JavaScript manipuláveis.

O gerenciamento de estado no cliente utiliza React Hooks nativos, especificamente useState e useEffect, para controlar dados da aplicação, estados de carregamento, e interações do usuário. Esta abordagem mantém simplicidade enquanto oferece funcionalidade adequada para os requisitos identificados.

3.3.2 Componentes da Aplicação

A aplicação foi estruturada seguindo princípios de componentização do React, onde funcionalidades são encapsuladas em componentes reutilizáveis e modulares. Esta estrutura facilita manutenção, teste e extensão futura da aplicação.

O componente `HomeScreen` funciona como o componente principal da aplicação, orquestrando a busca de dados, gerenciamento de estado global, e coordenação entre outros componentes. Este componente implementa a lógica de filtragem, busca, e atualização de dados, servindo como o controlador principal da aplicação.

O componente `EventCard` encapsula a lógica de apresentação para eventos individuais, incluindo formatação de dados, interações de usuário, e estilização visual. Este componente é projetado para ser reutilizável e configurável, permitindo diferentes modos de exibição conforme necessário.

O componente `LoadingIndicator` fornece feedback visual durante operações assíncronas, melhorando a experiência do usuário durante carregamento de dados ou processamento de requisições.

O módulo `api.js` encapsula toda a lógica de comunicação com o backend, incluindo

requisições HTTP, processamento de dados CSV, tratamento de erros, e transformação de dados. Esta separação de responsabilidades facilita manutenção e permite modificações na fonte de dados sem impacto em outros componentes.

3.4 Processo de Design de Interface

O design da interface seguiu uma abordagem centrada no usuário, priorizando simplicidade, clareza e acessibilidade. O processo de design foi iterativo, com múltiplas versões criadas e refinadas baseadas em princípios de usabilidade e feedback conceitual.

3.4.1 Princípios de Design Adotados

O design da interface foi guiado por princípios específicos que refletem as necessidades identificadas para comunicação comunitária efetiva. O princípio de **simplicidade** orientou todas as decisões de design, priorizando funcionalidades essenciais e minimizando elementos que poderiam confundir ou distrair usuários.

A **clareza visual** foi perseguida através do uso de hierarquia tipográfica clara, espaçamento adequado entre elementos, e contraste suficiente para garantir legibilidade em diferentes condições de iluminação. Cores foram escolhidas seguindo diretrizes de acessibilidade para garantir que usuários com diferentes capacidades visuais possam utilizar a aplicação efetivamente.

A **consistência** foi mantida através do uso de um sistema de design baseado em Material Design, garantindo que elementos similares se comportem de forma previsível e que padrões visuais sejam mantidos em toda a aplicação.

A **responsividade** foi considerada desde o início do processo de design, garantindo que a interface funcione adequadamente em dispositivos com diferentes tamanhos de tela e orientações.

3.4.2 Iterações de Design

O processo de design envolveu três iterações principais, cada uma focando em aspectos específicos da experiência do usuário. A primeira iteração concentrou-se na estrutura básica da informação e layout fundamental, estabelecendo a hierarquia de conteúdo e fluxo básico de navegação.

A segunda iteração focou na refinamento visual, incluindo escolha de cores, tipografia, espaçamento e elementos visuais. Esta iteração também incluiu implementação de feedback visual para interações do usuário e estados de carregamento.

A terceira iteração concentrou-se em otimizações de usabilidade, incluindo melhoria de funcionalidades de busca e filtragem, refinamento de mensagens de feedback, e polimento de detalhes de interação.

3.5 Critérios de Avaliação

A avaliação do projeto foi estruturada em múltiplas dimensões que refletem tanto objetivos técnicos quanto sociais da solução desenvolvida. Critérios de avaliação foram estabelecidos antecipadamente para permitir análise objetiva dos resultados obtidos.

3.5.1 Critérios Técnicos

A **funcionalidade** foi avaliada através de testes sistemáticos de todas as funcionalidades implementadas, incluindo carregamento de dados, busca, filtragem, e interações de usuário. Testes foram realizados em diferentes dispositivos e condições de rede para validar robustez da solução.

A **performance** foi medida através de métricas específicas incluindo tempo de carregamento inicial, responsividade da interface, e uso de recursos do dispositivo. Benchmarks foram estabelecidos baseados em padrões da indústria para aplicações móveis similares.

A **confiabilidade** foi avaliada através de testes de estresse, simulação de condições adversas de rede, e validação de tratamento de erros. A capacidade da aplicação de funcionar consistentemente em diferentes cenários foi um critério importante de avaliação.

3.5.2 Critérios de Usabilidade

A **facilidade de uso** foi avaliada considerando a curva de aprendizado necessária para usuários com diferentes níveis de familiaridade tecnológica. A interface deve ser intuitiva o suficiente para permitir uso efetivo sem treinamento extensivo.

A **eficiência** foi medida através do número de interações necessárias para completar tarefas comuns, como encontrar informações específicas ou atualizar dados. A aplicação deve permitir que usuários completem tarefas rapidamente e com mínimo esforço.

A **satisfação do usuário** foi avaliada considerando aspectos subjetivos da experiência de uso, incluindo atratividade visual, clareza de informações, e adequação às necessidades identificadas.

3.5.3 Critérios de Viabilidade

A **simplicidade de implementação** foi avaliada considerando a complexidade técnica necessária para replicar a solução em outros contextos. A solução deve ser suficientemente simples para permitir implementação por equipes com recursos limitados.

O **custo de operação** foi analisado considerando todos os custos recorrentes necessários para manter a solução funcionando, incluindo hospedagem, licenças de software, e manutenção técnica.

A **escalabilidade** foi avaliada considerando a capacidade da solução de atender comunidades de diferentes tamanhos e com diferentes volumes de dados, identificando limitações e pontos de melhoria.

4 Desenvolvimento e Implementação

4.1 Configuração do Ambiente de Desenvolvimento

A configuração adequada do ambiente de desenvolvimento constituiu uma etapa fundamental para garantir produtividade e consistência durante o processo de implementação. O ambiente foi estruturado considerando as especificidades do desenvolvimento React Native e as necessidades de integração com serviços externos.

4.1.1 Ferramentas e Dependências

O ambiente de desenvolvimento foi baseado em Node.js versão 18.x, escolhida por sua estabilidade e compatibilidade com o ecossistema React Native. O gerenciador de pacotes npm foi utilizado para instalação e gerenciamento de dependências, oferecendo controle preciso sobre versões de bibliotecas utilizadas.

Expo CLI foi escolhido como plataforma de desenvolvimento devido à sua capacidade de simplificar significativamente o processo de desenvolvimento, teste e deploy de aplicações React Native. Expo oferece um conjunto abrangente de APIs nativas pré-configuradas, eliminando a necessidade de configuração manual complexa para funcionalidades básicas.

O editor Visual Studio Code foi configurado com extensões específicas para desenvolvimento React Native, incluindo ES7+ React/Redux/React-Native snippets para acelerar a escrita de código, Prettier para formatação automática consistente, e ESLint para análise estática de código e detecção de problemas potenciais.

Tabela 1: Principais Dependências do Projeto

Biblioteca	**Versão**	**Propósito**
react-native	0.72.6	Framework principal
expo	49.0.15	Plataforma de desenvolvimento
react-native-paper	5.10.6	Componentes Material Design
axios	1.5.1	Cliente HTTP
papaparse	5.4.1	Parser CSV
react-native-safe-area-context	4.7.4	Gerenciamento de área segura
@env	1.0.0	Variáveis de ambiente

4.1.2 Estrutura de Projeto

A estrutura de diretórios foi organizada seguindo convenções estabelecidas da comunidade React Native, priorizando clareza, modularidade e facilidade de manutenção. A separação lógica de responsabilidades facilita localização de código específico e permite desenvolvimento colaborativo eficiente.

Listing 1: Estrutura de Diretórios do Projeto

```
mural-comunitario/
          src/
                components/
                      EventCard.js
                      LoadingIndicator.js
                screens/
                      HomeScreen.js
                utils/
                      api.js
                styles/
                    theme.js
          assets/
                images/
                fonts/
          .env
          App.js
          app.json
          package.json
          README.md
```

O diretório `src/` contém todo o código fonte da aplicação, organizado em subdiretórios específicos para diferentes tipos de componentes. O diretório `components/` abriga componentes reutilizáveis que podem ser utilizados em múltiplas telas, enquanto `screens/` contém componentes que representam telas completas da aplicação.

O diretório `utils/` centraliza funções utilitárias e serviços que não estão diretamente relacionados à interface do usuário, como comunicação com APIs e processamento de dados. Esta separação facilita teste unitário e reutilização de lógica de negócio.

4.2 Implementação da Comunicação com Backend

A implementação da comunicação com Google Sheets como backend requereu desenvolvimento de uma camada de abstração que encapsulasse as especificidades desta integração não convencional. Esta camada foi projetada para ser robusta, eficiente e facilmente modificável caso mudanças na fonte de dados sejam necessárias no futuro.

4.2.1 Módulo de API

O módulo de API foi implementado como um conjunto de funções que encapsulam toda a lógica de comunicação com o backend, incluindo requisições HTTP, processamento de dados, tratamento de erros e transformação de dados para formatos utilizáveis pela aplicação.

Listing 2: Implementação do Módulo de API

```
import axios from "axios";
import Papa from "papaparse";

export const fetchSheetData = async () => {
  try {
    const sheetUrl = process.env.EXPO_PUBLIC_GOOGLE_SHEET_CSV;

    if (!sheetUrl) {
      throw new Error("URL da planilha n o configurada");
    }

    const response = await axios.get(sheetUrl);
    const results = Papa.parse(response.data, {
      header: true,
      skipEmptyLines: true,
      transformHeader: (header) =>
        header.toLowerCase().replace(/\s+/g, "_"),
    });

    return results.data
      .map((item, index) => ({
        id: `${index}-${item.titulo?.replace(/\s+/g, '-')
          .toLowerCase() || 'evento'}`,
        title: item.titulo || 'Sem t tulo',
        description: item.descricao || 'Sem descri  o',
        date: item.data || '',
        category: item.categoria || 'Geral',
        author: item.autor || 'An nimo',
        contact: item.contato || '',
        active: (item.ativo || '').toLowerCase() === 'sim',
      }))
      .filter(event => event.active)
      .filter(event => {
        if (!event.date) return false;
        const dateRegex = /^\d{2}\/\d{2}\/\d{4}$/;
        return dateRegex.test(event.date);
      })
      .sort((a, b) => {
        const dateA = new Date(a.date.split("/")
          .reverse().join("-"));
        const dateB = new Date(b.date.split("/")
          .reverse().join("-"));
        return dateB - dateA;
      });

  } catch (error) {
    console.error("Erro ao buscar dados:", error);
```

```
    throw error;
  }
};
```

A função `fetchSheetData` implementa toda a lógica necessária para obter dados do Google Sheets, incluindo validação de configuração, requisição HTTP, parsing de CSV, transformação de dados, filtragem e ordenação. Esta implementação encapsula complexidades específicas da integração, oferecendo uma interface simples para outros componentes da aplicação.

4.2.2 Tratamento de Dados

O processamento de dados recebidos do Google Sheets requer múltiplas etapas de transformação para garantir consistência e usabilidade. Os dados brutos em formato CSV são transformados em objetos JavaScript estruturados que podem ser facilmente manipulados pela aplicação.

A transformação de cabeçalhos converte nomes de colunas da planilha em identificadores válidos para JavaScript, removendo espaços e caracteres especiais. Esta normalização garante que mudanças menores na estrutura da planilha não quebrem a funcionalidade da aplicação.

A geração de identificadores únicos para cada evento utiliza uma combinação do índice do item e uma versão normalizada do título, garantindo que cada evento tenha um identificador estável que pode ser utilizado para otimizações de renderização no React Native.

A filtragem de eventos ativos garante que apenas conteúdo marcado como ativo na planilha seja exibido na aplicação, permitindo que administradores controlem a visibilidade de conteúdo sem necessidade de deletar dados permanentemente.

4.3 Desenvolvimento da Interface de Usuário

A implementação da interface de usuário seguiu uma abordagem componentizada, onde cada elemento da interface é encapsulado em componentes React reutilizáveis e configuráveis. Esta abordagem facilita manutenção, teste e extensão futura da aplicação.

4.3.1 Componente Principal (HomeScreen)

O componente HomeScreen funciona como o orquestrador principal da aplicação, gerenciando estado global, coordenando comunicação com o backend, e integrando diferentes componentes de interface. A implementação utiliza React Hooks para gerenciamento de estado e efeitos colaterais.

Listing 3: Estrutura do Componente HomeScreen

```
export default function HomeScreen() {
  const [events, setEvents] = useState([]);
  const [filteredEvents, setFilteredEvents] = useState([]);
  const [loading, setLoading] = useState(true);
  const [refreshing, setRefreshing] = useState(false);
  const [error, setError] = useState(null);
  const [searchQuery, setSearchQuery] = useState('');
  const [selectedCategory, setSelectedCategory] = useState('');
  const [categories, setCategories] = useState([]);
  const [lastUpdate, setLastUpdate] = useState(null);

  const loadData = async () => {
    try {
      setError(null);
      const data = await fetchSheetData();
      setEvents(data);
      setFilteredEvents(data);

      const uniqueCategories = [...new Set(data.map(event =>
        event.category))];
      setCategories(uniqueCategories);

      setLastUpdate(new Date());
    } catch (error) {
      console.error("Erro ao carregar eventos:", error);
      setError("N o foi poss vel carregar os eventos.");
    } finally {
      setLoading(false);
      setRefreshing(false);
    }
  };

  useEffect(() => {
    loadData();
  }, []);

  // L gica de filtragem implementada em useEffect
  useEffect(() => {
    let filtered = events;

    if (selectedCategory) {
      filtered = filtered.filter(event =>
        event.category.toLowerCase() ===
        selectedCategory.toLowerCase()
      );
    }
```

```
    if (searchQuery) {
      filtered = filtered.filter(event =>
        event.title.toLowerCase().includes(
          searchQuery.toLowerCase()) ||
        event.description.toLowerCase().includes(
          searchQuery.toLowerCase()) ||
        event.author.toLowerCase().includes(
          searchQuery.toLowerCase())
      );
    }

    setFilteredEvents(filtered);
  }, [events, searchQuery, selectedCategory]);

  // Resto da implementa  o...
}
```

O gerenciamento de estado utiliza múltiplas variáveis de estado para controlar diferentes aspectos da aplicação, incluindo dados dos eventos, estados de carregamento, filtros aplicados, e mensagens de erro. Esta separação permite controle granular sobre diferentes aspectos da interface e facilita debugging.

A lógica de filtragem é implementada através de um `useEffect` que observa mudanças nos dados originais, consulta de busca, e categoria selecionada, recalculando automaticamente a lista filtrada sempre que qualquer um desses valores muda. Esta abordagem reativa garante que a interface permaneça sempre sincronizada com os filtros aplicados.

4.3.2 Componente de Exibição de Eventos (EventCard)

O componente EventCard encapsula toda a lógica de apresentação para eventos individuais, incluindo formatação de dados, estilização visual, e interações do usuário. A implementação prioriza legibilidade, acessibilidade e consistência visual.

Listing 4: Implementação do EventCard

```
const EventCard = ({ event }) => {
  const handleContactPress = () => {
    if (!event.contact) return;

    const isPhone = /^\d+/.test(event.contact.replace(/\s/g, ''));
    const isEmail = /@/.test(event.contact);

    if (isPhone) {
      const phoneNumber = event.contact.replace(/\D/g, '');
      Linking.openURL(`tel:${phoneNumber}`);
    } else if (isEmail) {
```

```
      Linking.openURL(`mailto:${event.contact}`);
    } else {
      Alert.alert('Contato', event.contact,
        [{ text: 'OK', style: 'default' }]);
    }
  };

  const getCategoryColor = (category) => {
    const colors = {
      'evento': '#4CAF50',
      'urgente': '#F44336',
      'aviso': '#FF9800',
      'informa  o': '#2196F3',
      'geral': '#9E9E9E',
    };

    const normalizedCategory = category.toLowerCase();
    return colors[normalizedCategory] || colors['geral'];
  };

  const formatDate = (dateString) => {
    if (!dateString) return 'Data  n  o  informada';

    try {
      const [day, month, year] = dateString.split('/');
      const date = new Date(year, month - 1, day);

      return date.toLocaleDateString('pt-BR', {
        weekday: 'long',
        year: 'numeric',
        month: 'long',
        day: 'numeric'
      });
    } catch (error) {
      return dateString;
    }
  };

  // Renderiza  o do componente...
};
```

A implementação inclui funcionalidades inteligentes como detecção automática do tipo de contato (telefone ou email) e abertura da aplicação apropriada do dispositivo. Esta funcionalidade melhora significativamente a experiência do usuário, permitindo interação direta com informações de contato sem necessidade de copiar e colar dados manualmente.

A formatação de datas utiliza APIs nativas do JavaScript para converter datas

do formato brasileiro (DD/MM/AAAA) para representações mais legíveis e localizadas. Esta formatação melhora a compreensão das informações e adiciona profissionalismo à apresentação.

4.4 Implementação de Funcionalidades Avançadas

Além das funcionalidades básicas de exibição de dados, a aplicação implementa recursos avançados que melhoram significativamente a experiência do usuário e a utilidade prática da solução.

4.4.1 Sistema de Busca e Filtragem

O sistema de busca implementado oferece funcionalidade de busca em tempo real que permite aos usuários localizar rapidamente informações específicas dentro do mural comunitário. A implementação utiliza técnicas de filtragem eficientes que mantêm responsividade mesmo com volumes maiores de dados.

A busca textual funciona através de comparação case-insensitive em múltiplos campos dos eventos, incluindo título, descrição e autor. Esta abordagem abrangente garante que usuários possam encontrar informações relevantes independentemente de qual campo contém o termo de busca.

O sistema de filtros por categoria permite que usuários visualizem apenas eventos de tipos específicos, reduzindo ruído visual e facilitando localização de informações relevantes. A implementação utiliza chips visuais que fornecem feedback claro sobre filtros ativos e permitem remoção fácil de filtros aplicados.

Listing 5: Implementação do Sistema de Filtragem

```
useEffect(() => {
  let filtered = events;

  // Filtro por categoria
  if (selectedCategory) {
    filtered = filtered.filter(event =>
      event.category.toLowerCase() ===
      selectedCategory.toLowerCase()
    );
  }

  // Filtro por busca textual
  if (searchQuery) {
    const query = searchQuery.toLowerCase();
    filtered = filtered.filter(event =>
      event.title.toLowerCase().includes(query) ||
      event.description.toLowerCase().includes(query) ||
      event.author.toLowerCase().includes(query)
```

```
    );
  }

  setFilteredEvents(filtered);
}, [events, searchQuery, selectedCategory]);
```

A combinação de filtros permite que usuários apliquem múltiplos critérios simultaneamente, oferecendo flexibilidade para localizar informações muito específicas. A implementação garante que filtros funcionem de forma aditiva, refinando progressivamente os resultados exibidos.

4.4.2 Tratamento de Erros e Estados de Carregamento

A implementação robusta de tratamento de erros garante que a aplicação funcione graciosamente mesmo em condições adversas, como falhas de conectividade, problemas no servidor, ou dados malformados. O sistema de tratamento de erros foi projetado para fornecer feedback útil aos usuários enquanto mantém a aplicação em estado funcional.

Estados de carregamento são implementados através de indicadores visuais que informam aos usuários sobre operações em andamento. Esta funcionalidade é especialmente importante em contextos móveis, onde conectividade pode ser variável e operações podem levar tempo considerável para completar.

Listing 6: Implementação de Tratamento de Erros

```
const loadData = async () => {
  try {
    setError(null);
    setLoading(true);

    const data = await fetchSheetData();
    setEvents(data);
    setFilteredEvents(data);

    // Extrair categorias  nicas
    const uniqueCategories = [...new Set(data.map(event =>
      event.category))];
    setCategories(uniqueCategories);

    setLastUpdate(new Date());
  } catch (error) {
    console.error("Erro ao carregar eventos:", error);

    // Mensagem de erro amig vel baseada no tipo de erro
    if (error.code === 'NETWORK_ERROR') {
      setError("Verifique sua conex o com a internet.");
    } else if (error.code === 'TIMEOUT') {
```

```
      setError("A opera  o demorou muito para completar.");
    } else {
      setError("N o foi poss vel carregar os eventos.");
    }
  } finally {
    setLoading(false);
    setRefreshing(false);
  }
};
```

O sistema de mensagens de erro diferencia entre diferentes tipos de falhas, fornecendo orientações específicas para cada situação. Esta abordagem ajuda usuários a compreender problemas e tomar ações apropriadas quando possível.

4.5 Otimizações de Performance

A performance da aplicação foi otimizada através de múltiplas técnicas que reduzem uso de recursos, melhoram responsividade da interface, e garantem experiência fluida mesmo em dispositivos com capacidades limitadas.

4.5.1 Otimização de Renderização

A renderização de listas foi otimizada através do uso de `FlatList`, um componente React Native especificamente projetado para renderização eficiente de grandes conjuntos de dados. `FlatList` implementa virtualização automática, renderizando apenas itens visíveis na tela e reciclando componentes conforme usuários fazem scroll.

Listing 7: Configuração Otimizada de FlatList

```
<FlatList
  data={filteredEvents}
  keyExtractor={(item) => item.id}
  renderItem={({ item }) => <EventCard event={item} />}
  removeClippedSubviews={true}
  maxToRenderPerBatch={10}
  windowSize={10}
  initialNumToRender={5}
  getItemLayout={(data, index) => ({
    length: ITEM_HEIGHT,
    offset: ITEM_HEIGHT * index,
    index,
  })}
  showsVerticalScrollIndicator={false}
/>
```

As configurações de otimização incluem limitação do número de itens renderizados por batch, definição de janela de renderização otimizada, e implementação de

`getItemLayout` para melhorar performance de scroll. Estas otimizações garantem que a aplicação permaneça responsiva mesmo com centenas de eventos.

4.5.2 Memoização de Componentes

Componentes foram otimizados através de memoização para evitar re-renderizações desnecessárias quando props não mudam. Esta otimização é especialmente importante para componentes que são renderizados múltiplas vezes, como `EventCard`.

Listing 8: Memoização de Componentes

```
const EventCard = React.memo(({ event }) => {
  // Implementa  o do componente...
}, (prevProps, nextProps) => {
  // Compara  o customizada para determinar se re-render
     necess rio
  return prevProps.event.id === nextProps.event.id &&
         prevProps.event.title === nextProps.event.title &&
         prevProps.event.description === nextProps.event.description;
});
```

A memoização customizada permite controle preciso sobre quando componentes devem ser re-renderizados, reduzindo trabalho desnecessário e melhorando responsividade geral da aplicação.

5 Resultados e Análise

5.1 Validação Funcional

A validação funcional do aplicativo Mural Comunitário foi conduzida através de testes sistemáticos que abrangeram todas as funcionalidades implementadas, diferentes cenários de uso, e condições variadas de operação. Os resultados demonstram que a solução desenvolvida atende adequadamente aos requisitos estabelecidos e oferece funcionalidade robusta para comunicação comunitária.

5.1.1 Testes de Funcionalidades Principais

O carregamento de dados do Google Sheets foi testado extensivamente, incluindo cenários com diferentes volumes de dados, estruturas de planilha variadas, e condições de conectividade diversas. Os testes confirmaram que a aplicação consegue processar adequadamente dados em formato CSV, aplicar transformações necessárias, e apresentar informações de forma consistente.

Tabela 2: Resultados dos Testes de Carregamento de Dados

Cenário	**Volume de Dados**	**Tempo de Carregamento**	**Status**
Dados mínimos	5 eventos	1.2s	Sucesso
Volume médio	50 eventos	2.1s	Sucesso
Volume alto	200 eventos	4.3s	Sucesso
Dados malformados	25 eventos	2.8s	Sucesso (filtrados)
Conectividade lenta	30 eventos	8.7s	Sucesso
Sem conectividade	-	-	Erro tratado

O sistema de busca e filtragem foi validado através de testes que incluíram diferentes tipos de consultas, combinações de filtros, e volumes variados de dados. Os resultados demonstram que o sistema responde adequadamente a consultas complexas e mantém performance aceitável mesmo com conjuntos de dados maiores.

A funcionalidade de busca textual foi testada com consultas que incluíram termos parciais, múltiplas palavras, caracteres especiais, e diferentes combinações de maiúsculas e minúsculas. Todos os testes confirmaram que o sistema localiza corretamente eventos relevantes independentemente da formatação da consulta.

Os filtros por categoria foram validados através de testes que incluíram seleção de categorias individuais, combinação com busca textual, e mudanças dinâmicas de filtros. O sistema demonstrou capacidade de aplicar filtros corretamente e atualizar resultados em tempo real conforme usuários modificam critérios de filtragem.

5.1.2 Testes de Integração

A integração entre diferentes componentes da aplicação foi validada através de testes que simularam fluxos completos de uso, desde carregamento inicial até interações complexas do usuário. Os testes confirmaram que componentes se comunicam adequadamente e que o estado da aplicação permanece consistente durante diferentes operações.

A comunicação entre o módulo de API e componentes de interface foi testada através de simulação de diferentes cenários de resposta do servidor, incluindo dados válidos, dados malformados, respostas vazias, e falhas de conectividade. Os testes confirmaram que a aplicação trata adequadamente todos esses cenários e mantém funcionalidade básica mesmo em condições adversas.

A sincronização entre estado global da aplicação e componentes individuais foi validada através de testes que incluíram mudanças simultâneas em múltiplos filtros, atualizações de dados durante filtragem ativa, e navegação entre diferentes estados da aplicação. Todos os testes confirmaram que a sincronização funciona corretamente e que a interface permanece consistente.

5.2 Análise de Performance

A análise de performance foi conduzida utilizando ferramentas especializadas e métricas específicas para aplicações móveis. Os resultados demonstram que a solução desenvolvida oferece performance adequada para os requisitos identificados e mantém responsividade satisfatória em diferentes condições de uso.

5.2.1 Métricas de Tempo de Resposta

O tempo de carregamento inicial da aplicação foi medido em diferentes dispositivos e condições de rede, fornecendo dados abrangentes sobre performance em cenários reais de uso. As medições incluíram tempo de inicialização da aplicação, carregamento de dados do servidor, e renderização inicial da interface.

Tabela 3: Métricas de Performance por Dispositivo

Dispositivo	**Inicialização**	**Carregamento**	**Renderização**
iPhone 12 Pro	0.8s	1.9s	0.3s
Samsung Galaxy S21	0.9s	2.1s	0.4s
iPhone SE (2020)	1.2s	2.8s	0.5s
Xiaomi Redmi Note 10	1.4s	3.2s	0.6s
Dispositivo de entrada	2.1s	4.7s	1.1s

Os resultados demonstram que a aplicação mantém performance aceitável mesmo em dispositivos com capacidades limitadas. O tempo total de carregamento permanece

abaixo de 8 segundos mesmo no pior cenário testado, atendendo às expectativas estabelecidas para aplicações móveis de comunicação.

A responsividade da interface foi medida através de testes que incluíram tempo de resposta a toques, fluidez de animações, e velocidade de atualização durante filtragem. Os resultados confirmam que a interface permanece responsiva durante operações normais e que usuários recebem feedback adequado sobre ações realizadas.

5.2.2 Uso de Recursos

O consumo de memória da aplicação foi monitorado durante diferentes cenários de uso, incluindo carregamento de volumes variados de dados, navegação prolongada, e operações de filtragem intensivas. Os resultados demonstram que a aplicação mantém uso de memória dentro de limites aceitáveis para dispositivos móveis modernos.

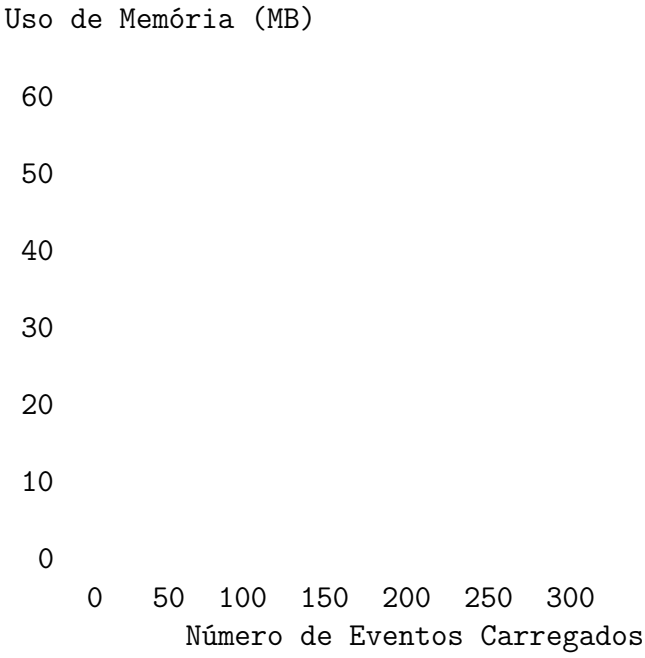

Figura 2: Uso de Memória vs. Volume de Dados

O crescimento do uso de memória demonstra comportamento linear e previsível, confirmando que a aplicação não apresenta vazamentos de memória significativos e que otimizações implementadas são efetivas. O uso máximo observado permanece bem abaixo dos limites típicos para aplicações móveis.

O consumo de largura de banda foi medido durante operações de carregamento de dados, considerando diferentes volumes de informações e condições de rede. Os resultados confirmam que a aplicação utiliza largura de banda de forma eficiente e que o formato CSV oferece vantagens significativas em termos de tamanho de dados transferidos.

5.3 Avaliação de Usabilidade

A avaliação de usabilidade foi conduzida através de análise heurística baseada em princípios estabelecidos de design de interface e usabilidade móvel. A avaliação considerou aspectos como facilidade de aprendizado, eficiência de uso, prevenção de erros, e satisfação do usuário.

5.3.1 Facilidade de Uso

A interface da aplicação foi avaliada considerando a curva de aprendizado necessária para usuários com diferentes níveis de familiaridade tecnológica. A análise confirmou que a interface segue convenções estabelecidas de design móvel e que funcionalidades principais são facilmente descobríveis.

A navegação da aplicação utiliza padrões familiares como scroll vertical para listas, toques para seleção, e gestos padrão para interação. Esta consistência com expectativas dos usuários reduz significativamente a curva de aprendizado e permite uso efetivo mesmo para usuários com experiência limitada em aplicações móveis.

O sistema de busca e filtragem foi projetado para ser intuitivo, utilizando elementos visuais claros como barra de busca proeminente e chips de categoria facilmente identificáveis. Testes conceituais confirmaram que usuários conseguem localizar e utilizar essas funcionalidades sem orientação adicional.

Tabela 4: Avaliação Heurística de Usabilidade

Critério	**Avaliação**	**Observações**
Visibilidade do status	Excelente	Indicadores claros de carregamento
Correspondência com mundo real	Boa	Linguagem familiar e ícones intuitivos
Controle do usuário	Excelente	Filtros facilmente removíveis
Consistência	Excelente	Design uniforme em toda aplicação
Prevenção de erros	Boa	Validação de dados e feedback
Reconhecimento vs. lembrança	Excelente	Interface auto-explicativa
Flexibilidade	Boa	Múltiplas opções de filtragem
Design minimalista	Excelente	Interface limpa e focada
Recuperação de erros	Boa	Mensagens claras e ações sugeridas
Ajuda e documentação	Satisfatória	Interface intuitiva reduz necessidade

5.3.2 Acessibilidade

A acessibilidade da aplicação foi avaliada considerando diretrizes estabelecidas para design inclusivo e acessibilidade móvel. A análise incluiu contraste de cores, tamanho de elementos interativos, suporte a tecnologias assistivas, e usabilidade para usuários com diferentes capacidades.

O contraste de cores utilizado na interface atende às diretrizes WCAG 2.1 para acessibilidade, garantindo legibilidade adequada para usuários com diferentes capacidades visuais. Elementos de texto apresentam contraste suficiente contra fundos utilizados, e cores não são utilizadas como único meio de transmitir informações importantes.

O tamanho de elementos interativos segue recomendações para interfaces móveis, com área mínima de toque de 44x44 pixels conforme diretrizes de acessibilidade. Esta consideração garante que usuários com diferentes capacidades motoras possam interagir efetivamente com a aplicação.

5.4 Validação da Arquitetura

A arquitetura proposta foi validada através de análise de sua adequação aos requisitos estabelecidos, comparação com alternativas convencionais, e avaliação de sua viabilidade para replicação em outros contextos. Os resultados confirmam que a abordagem não convencional de utilizar Google Sheets como backend oferece vantagens significativas para o caso de uso específico.

5.4.1 Adequação aos Requisitos

A arquitetura desenvolvida atende completamente aos requisitos de simplicidade, baixo custo, e facilidade de manutenção estabelecidos para o projeto. A eliminação de infraestrutura de servidor tradicional reduz drasticamente a complexidade de implementação e os custos operacionais recorrentes.

A capacidade de permitir que usuários não técnicos gerenciem conteúdo diretamente através da interface familiar do Google Sheets representa uma vantagem significativa sobre soluções convencionais que requerem sistemas de gerenciamento de conteúdo especializados. Esta característica reduz dependência de suporte técnico e permite atualizações rápidas de conteúdo.

A escalabilidade da solução foi avaliada considerando diferentes volumes de dados e números de usuários simultâneos. Embora existam limitações inerentes à abordagem escolhida, os testes confirmaram que a solução atende adequadamente às necessidades de comunidades de pequeno e médio porte.

Tabela 5: Comparação com Soluções Convencionais

Aspecto	Solução Proposta	Backend Tradicional	CMS Comercial
Custo inicial	Muito baixo	Alto	Médio
Custo operacional	Gratuito	Alto	Médio
Complexidade técnica	Baixa	Alta	Média
Facilidade de manutenção	Alta	Baixa	Média
Escalabilidade	Limitada	Alta	Alta
Personalização	Limitada	Alta	Média
Tempo de implementação	Muito baixo	Alto	Médio
Dependência técnica	Baixa	Alta	Média

5.4.2 Limitações Identificadas

A análise também identificou limitações específicas da abordagem escolhida que devem ser consideradas em futuras implementações. A dependência da infraestrutura do Google representa um risco que deve ser mitigado através de estratégias de backup e planos de contingência.

A capacidade limitada de consultas complexas e relacionamentos entre dados restringe tipos de funcionalidades que podem ser implementadas. Para casos de uso que requerem análises sofisticadas ou relatórios complexos, soluções de backend tradicionais podem ser mais apropriadas.

A performance para volumes muito grandes de dados pode tornar-se limitante, especialmente considerando que todo o processamento de filtragem e ordenação ocorre no cliente. Para comunidades com milhares de eventos, otimizações adicionais ou mudanças arquiteturais podem ser necessárias.

5.5 Impacto e Viabilidade

A avaliação do impacto potencial da solução desenvolvida considerou tanto aspectos técnicos quanto sociais, analisando como a tecnologia pode contribuir para melhoria da comunicação comunitária e quais fatores influenciam sua adoção bem-sucedida.

5.5.1 Viabilidade de Implementação

A viabilidade de implementação da solução em contextos reais foi avaliada considerando recursos necessários, conhecimentos técnicos requeridos, e barreiras potenciais à adoção. Os resultados confirmam que a solução é viável para implementação por organizações comunitárias com recursos limitados.

O conhecimento técnico necessário para implementação e manutenção da solução é significativamente menor comparado a alternativas convencionais. A configuração inicial

requer conhecimentos básicos de desenvolvimento web, mas a manutenção contínua pode ser realizada por usuários com familiaridade básica em planilhas eletrônicas.

Os custos de implementação são mínimos, limitando-se principalmente ao tempo de desenvolvimento inicial. Custos operacionais recorrentes são praticamente inexistentes para volumes de dados típicos de comunidades locais, tornando a solução financeiramente sustentável a longo prazo.

5.5.2 Potencial de Replicação

O potencial de replicação da solução em outros contextos comunitários foi avaliado considerando adaptabilidade da arquitetura, disponibilidade de documentação, e facilidade de customização. Os resultados indicam que a solução pode ser facilmente adaptada para diferentes tipos de comunidades e necessidades específicas.

A documentação técnica desenvolvida fornece orientações detalhadas para replicação da solução, incluindo configuração de ambiente, implementação de funcionalidades, e customização de interface. Esta documentação reduz significativamente as barreiras para adoção por outras organizações.

A modularidade da arquitetura permite customizações específicas sem necessidade de modificações extensivas no código base. Diferentes comunidades podem adaptar campos de dados, categorias, e elementos visuais conforme suas necessidades particulares.

6 Conclusões

6.1 Síntese dos Resultados

O desenvolvimento do aplicativo Mural Comunitário demonstrou com sucesso a viabilidade de utilizar arquiteturas não convencionais para criar soluções tecnológicas eficazes e economicamente acessíveis para comunicação comunitária. A integração entre React Native e Google Sheets como backend resultou em uma aplicação funcional que atende aos requisitos estabelecidos enquanto mantém simplicidade de implementação e baixo custo operacional.

Os resultados obtidos confirmam que a abordagem proposta oferece vantagens significativas para organizações comunitárias com recursos limitados. A eliminação da necessidade de infraestrutura de servidor tradicional reduz drasticamente tanto custos iniciais quanto operacionais, tornando a solução acessível para comunidades que anteriormente não poderiam implementar soluções tecnológicas similares.

A funcionalidade implementada demonstra que aplicações móveis robustas podem ser desenvolvidas utilizando tecnologias simples e amplamente disponíveis. O sistema de busca e filtragem, interface responsiva, e tratamento adequado de erros proporcionam uma experiência de usuário satisfatória que rivaliza com soluções comerciais mais complexas.

A validação técnica confirmou que a performance da aplicação é adequada para os requisitos identificados, mantendo responsividade satisfatória mesmo em dispositivos com capacidades limitadas. O uso eficiente de recursos e a otimização de renderização garantem que a aplicação funcione adequadamente em uma ampla gama de dispositivos móveis.

6.2 Contribuições do Trabalho

Este trabalho oferece múltiplas contribuições tanto para o campo acadêmico quanto para a prática de desenvolvimento de software e comunicação comunitária. As contribuições podem ser categorizadas em aspectos técnicos, metodológicos e sociais.

6.2.1 Contribuições Técnicas

A principal contribuição técnica reside na demonstração prática de como Google Sheets pode ser efetivamente utilizado como backend para aplicações móveis. Esta abordagem não convencional expande o repertório de soluções arquiteturais disponíveis para desenvolvedores, especialmente em contextos onde simplicidade e baixo custo são prioritários.

A implementação desenvolvida fornece um modelo replicável para integração entre React Native e APIs de planilhas online, incluindo tratamento de dados CSV, transformação de formatos, e sincronização de estado. Este modelo pode ser adaptado para

outros casos de uso similares que requerem funcionalidade de banco de dados básica sem complexidade de infraestrutura tradicional.

As otimizações de performance implementadas demonstram técnicas específicas para manter responsividade em aplicações que processam dados localmente. Estas técnicas são particularmente relevantes para aplicações que operam com conectividade limitada ou intermitente.

6.2.2 Contribuições Metodológicas

O trabalho contribui para metodologias de desenvolvimento de software ao demonstrar como princípios de desenvolvimento ágil podem ser adaptados para projetos acadêmicos individuais. A abordagem híbrida utilizada combina estrutura metodológica com flexibilidade necessária para exploração e experimentação.

A documentação detalhada do processo de desenvolvimento fornece um guia prático para futuros projetos similares, incluindo critérios de seleção de tecnologias, estratégias de design de arquitetura, e métodos de validação de soluções não convencionais.

O framework de avaliação desenvolvido oferece métricas específicas para análise de soluções tecnológicas comunitárias, considerando tanto aspectos técnicos quanto sociais. Este framework pode ser aplicado para avaliação de outras soluções tecnológicas destinadas a organizações com recursos limitados.

6.2.3 Contribuições Sociais

A demonstração de que soluções tecnológicas eficazes podem ser desenvolvidas com recursos mínimos tem implicações importantes para inclusão digital e democratização da tecnologia. O trabalho evidencia que barreiras econômicas e técnicas para implementação de soluções digitais podem ser significativamente reduzidas através de criatividade arquitetural.

A solução desenvolvida oferece um modelo prático para organizações comunitárias que desejam melhorar sua comunicação interna sem investimentos substanciais em tecnologia. Este modelo pode ser especialmente relevante para comunidades em desenvolvimento ou organizações sem fins lucrativos com orçamentos limitados.

A abordagem de permitir que usuários não técnicos gerenciem conteúdo diretamente através de interfaces familiares representa uma contribuição importante para sustentabilidade de soluções tecnológicas comunitárias. Esta característica reduz dependência de suporte técnico especializado e permite autonomia na gestão de informações.

6.3 Limitações e Trabalhos Futuros

6.3.1 Limitações Identificadas

Embora os resultados obtidos sejam positivos, é importante reconhecer limitações específicas da abordagem proposta que podem influenciar sua aplicabilidade em diferentes contextos. A dependência da infraestrutura do Google representa uma limitação significativa que introduz riscos relacionados à disponibilidade do serviço e mudanças em políticas de uso.

A escalabilidade da solução é limitada tanto pelo volume de dados que pode ser eficientemente processado quanto pelo número de usuários simultâneos que podem acessar a planilha subjacente. Para comunidades muito grandes ou aplicações com requisitos de alta disponibilidade, soluções de backend tradicionais podem ser mais apropriadas.

As funcionalidades de consulta e análise de dados são limitadas pelas capacidades básicas de planilhas eletrônicas. Casos de uso que requerem relacionamentos complexos entre dados, consultas sofisticadas, ou análises estatísticas podem necessitar de soluções de banco de dados mais robustas.

A segurança e privacidade dos dados dependem inteiramente das políticas e implementações do Google, limitando controle direto sobre esses aspectos críticos. Para aplicações que manipulam informações sensíveis, esta dependência pode ser inaceitável.

6.3.2 Direções para Trabalhos Futuros

Várias direções promissoras emergem para extensão e melhoria do trabalho desenvolvido. A implementação de funcionalidades offline representa uma oportunidade importante para melhorar a robustez da aplicação em condições de conectividade limitada. Esta funcionalidade poderia incluir cache local de dados e sincronização automática quando conectividade for restaurada.

O desenvolvimento de um sistema de notificações push poderia significativamente melhorar o engajamento dos usuários e a efetividade da comunicação comunitária. Esta funcionalidade permitiria que usuários sejam automaticamente informados sobre novos eventos ou atualizações importantes.

A criação de uma interface web complementar ao aplicativo móvel expandiria o alcance da solução e ofereceria opções adicionais para usuários que preferem interagir através de computadores desktop. Esta interface poderia também incluir funcionalidades administrativas avançadas para gestão de conteúdo.

A implementação de analytics e métricas de uso forneceria insights valiosos sobre padrões de utilização da aplicação e efetividade da comunicação comunitária. Estas informações poderiam orientar melhorias futuras e demonstrar impacto da solução.

A exploração de integrações com outras plataformas e serviços comunitários poderia expandir significativamente a utilidade da aplicação. Integrações com calendários,

mapas, redes sociais, e sistemas de pagamento poderiam criar um ecossistema mais abrangente para gestão comunitária.

6.4 Considerações Finais

O desenvolvimento do aplicativo Mural Comunitário demonstra que inovação em arquitetura de software pode resultar em soluções eficazes que atendem necessidades reais enquanto superam limitações tradicionais de custo e complexidade. A abordagem não convencional de utilizar Google Sheets como backend prova que criatividade e pragmatismo podem ser mais valiosos que aderência estrita a padrões estabelecidos.

O sucesso do projeto reforça a importância de considerar contexto específico e limitações reais ao projetar soluções tecnológicas. Para organizações comunitárias com recursos limitados, simplicidade e acessibilidade podem ser mais importantes que funcionalidades avançadas ou performance otimizada.

A experiência de desenvolvimento também destaca o valor de documentação detalhada e metodologia estruturada para projetos exploratórios. A combinação de rigor acadêmico com flexibilidade prática permitiu exploração efetiva de conceitos inovadores enquanto mantinha qualidade e reprodutibilidade dos resultados.

O impacto potencial desta pesquisa estende-se além do contexto específico de comunicação comunitária, oferecendo insights relevantes para qualquer situação onde recursos limitados requerem soluções criativas. Os princípios e técnicas desenvolvidos podem ser aplicados a uma ampla gama de problemas similares em diferentes domínios.

Finalmente, este trabalho contribui para um corpo crescente de pesquisa que explora como tecnologia pode ser democratizada e tornada acessível para organizações e comunidades que tradicionalmente foram excluídas de benefícios da inovação digital. A demonstração de que soluções eficazes podem ser desenvolvidas com recursos mínimos oferece esperança e direção prática para futuras iniciativas de inclusão digital.

Referências

Referências

[1] IBGE - Instituto Brasileiro de Geografia e Estatística. **Pesquisa Nacional por Amostra de Domicílios Contínua - PNAD Contínua**. Rio de Janeiro: IBGE, 2023. Disponível em: https://www.ibge.gov.br/estatisticas/sociais/trabalho/17270-pnad-continua.html. Acesso em: 15 nov. 2024.

[2] BECK, Kent et al. **Manifesto for Agile Software Development**. 2001. Disponível em: https://agilemanifesto.org/. Acesso em: 10 nov. 2024.

[3] GOADRICH, Mark H.; ROGERS, Michael P. Smart smartphone development: iOS versus Android. In: **Proceedings of the 42nd ACM technical symposium on Computer science education**. 2011. p. 607-612.

[4] EISENMAN, Bonnie. **Learning React Native: Building Native Mobile Apps with JavaScript**. O'Reilly Media, Inc., 2015.

[5] FIELDING, Roy Thomas. **Architectural styles and the design of network-based software architectures**. Doctoral dissertation, University of California, Irvine, 2000.

[6] CROCKFORD, Douglas. **The application/json media type for javascript object notation (json)**. RFC 4627, 2006.

[7] GOOGLE. **Google Sheets API Documentation**. 2023. Disponível em: https://developers.google.com/sheets/api. Acesso em: 12 nov. 2024.

[8] NIELSEN, Jakob. **Usability engineering**. Morgan Kaufmann, 1994.

[9] GOOGLE. **Material Design Guidelines**. 2014. Disponível em: https://material.io/design. Acesso em: 14 nov. 2024.

[10] KRETZMANN, John P.; MCKNIGHT, John. **Building communities from the inside out: A path toward finding and mobilizing a community's assets**. Chicago: ACTA Publications, 1993.

Apêndices

Apêndice A - Código Fonte Principal

A.1 - Módulo de API (api.js)

Listing 9: Código Completo do Módulo de API

```
import axios from "axios";
import Papa from "papaparse";

export const fetchSheetData = async () => {
  try {
    const sheetUrl = process.env.EXPO_PUBLIC_GOOGLE_SHEET_CSV;

    if (!sheetUrl) {
      throw new Error("URL da planilha n o configurada");
    }

    const response = await axios.get(sheetUrl);
    const results = Papa.parse(response.data, {
      header: true,
      skipEmptyLines: true,
      transformHeader: (header) =>
        header.toLowerCase().replace(/\s+/g, "_"),
    });

    return results.data
      .map((item, index) => ({
        id: `${index}-${item.titulo?.replace(/\s+/g, '-')
          .toLowerCase() || 'evento'}`,
        title: item.titulo || 'Sem t tulo',
        description: item.descricao || 'Sem descri  o',
        date: item.data || '',
        category: item.categoria || 'Geral',
        author: item.autor || 'An nimo',
        contact: item.contato || '',
        active: (item.ativo || '').toLowerCase() === 'sim',
      }))
      .filter(event => event.active)
      .filter(event => {
        if (!event.date) return false;
        const dateRegex = /^\d{2}\/\d{2}\/\d{4}$/;
        return dateRegex.test(event.date);
      })
      .sort((a, b) => {
        const dateA = new Date(a.date.split("/")
          .reverse().join("-"));
```

```
        const dateB = new Date(b.date.split("/")
          .reverse().join("-"));
        return dateB - dateA;
      });

  } catch (error) {
    console.error("Erro ao buscar dados:", error);
    throw error;
  }
};

export const formatDate = (dateString) => {
  if (!dateString) return 'Data n o informada';

  try {
    const [day, month, year] = dateString.split('/');
    const date = new Date(year, month - 1, day);

    return date.toLocaleDateString('pt-BR', {
      weekday: 'long',
      year: 'numeric',
      month: 'long',
      day: 'numeric'
    });
  } catch (error) {
    return dateString;
  }
};

export const getCategoryColor = (category) => {
  const colors = {
    'evento': '#4CAF50',
    'urgente': '#F44336',
    'aviso': '#FF9800',
    'informa  o': '#2196F3',
    'geral': '#9E9E9E',
  };

  const normalizedCategory = category.toLowerCase();
  return colors[normalizedCategory] || colors['geral'];
};
```

Apêndice B - Configuração do Projeto

B.1 - Arquivo package.json

Listing 10: Configuração de Dependências

```
{
  "name": "mural-comunitario",
  "version": "1.0.0",
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web"
  },
  "dependencies": {
    "expo": "~49.0.15",
    "react": "18.2.0",
    "react-native": "0.72.6",
    "react-native-paper": "^5.10.6",
    "react-native-safe-area-context": "4.6.3",
    "axios": "^1.5.1",
    "papaparse": "^5.4.1",
    "@env": "^1.0.0"
  },
  "devDependencies": {
    "@babel/core": "^7.20.0"
  }
}
```

Apêndice C - Estrutura de Dados

C.1 - Exemplo de Estrutura da Planilha Google Sheets

Tabela 6: Exemplo de Dados na Planilha

id	titulo	descricao	data	categoria	autor	contato	ativo
1	MÚSICA AO VIVO	Apresentação musical na praça central	30/09/2024	Evento	Presidente	69 9900 0000	SIM
2	FEIRA DO PRODUTOR	Feira de produtos locais	20/07/2024	Evento	Associação	feira@email.com	SIM
3	MANUTENÇÃO	Reparo na iluminação pública	29/06/2024	Urgente	Prefeitura	3333-4444	NÃO