UNESP - UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"

Faculdade de Ciências

Bacharelado em Sistemas de Informação

CHRISTIAN RODRIGUES BATISTA

COMUNIDADE COLO DE DEUS E SANTÍSSIMA VIRGEM: WEBSITE DE EVENTOS

BAURU - SP

2023

CHRISTIAN RODRIGUES BATISTA

COMUNIDADE COLO DE DEUS E SANTÍSSIMA VIRGEM: WEBSITE DE EVENTOS

Trabalho de Conclusão de Curso que discorre sobre a importância e necessidade da autonomia por meio de um website. Este é elaborado a partir das indicações das normas da Associação Brasileira de Normas Técnicas (ABNT) pelo aluno Christian Rodrigues Batista, matriculado no curso de Bacharelado de Sistemas de Informação, orientado pelo Prof. Dr. José Remo Ferreira Brega, sendo este documento material do processo para conclusão do curso.

BAURU - SP
2023

B333c

Batista, Christian Rodrigues
Comunidade Colo De Deus e Santíssima Virgem: Website de eventos / Christian Rodrigues Batista. -- Bauru, 2023
39 f. : il.

Trabalho de conclusão de curso (Bacharelado - Sistemas de Informação) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências, Bauru
Orientador: José Remo Ferreira Brega

1. Sistemas de computação. 2. Programação para internet. 3. Comunidades cristãs igreja católica. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

AGRADECIMENTOS

Primeiramente a Deus que permitiu que tudo isso acontecesse, ao longo de minha vida, e não somente nestes anos como universitário, mas que em todos os momentos é o maior mestre que alguém pode conhecer.

Agradeço a minha mãe Simone e meu Pai Sidney, heróis que me deram apoio, incentivo nas horas difíceis, de desânimo e cansaço. E a minha irmã Lívia, que nunca deixou de sorrir, transmitindo amor e felicidade para mim nos momentos turbulentos.

Agradecer especialmente a comunidade Colo de Deus que permitiu esse projeto ser construído e deu todo apoio necessário para a realização do mesmo. Principalmente ao Felipe Taborda, líder da missão Bauru - SP, e o Matheus Antunes Melo da frente celular de Londrina - PR, que foram via facilitadora de comunicação entre o núcleo da comunidade e eu.

Ao Prof. Dr. José Remo Ferreira Brega pela paciência e apoio na elaboração deste trabalho.

A Universidade Estadual Paulista "Júlio De Mesquita Filho", pela oportunidade de fazer o curso.

Quero agradecer também ao meu melhor amigo, William Paulo Gonçalves Braz, graduado em Arquitetura pela USP de São Carlos. Obrigado pelas palavras motivacionais, puxões de orelha e por me acolher. As risadas que compartilhei estão gravadas na memória. Obrigado por tudo.

RESUMO

Certamente, um website é uma ferramenta essencial na era digital e pode ser criado com diversas tecnologias disponíveis no mercado, como o React, uma biblioteca JavaScript popular e eficiente para a construção de interfaces de usuário. Com essa tecnologia, foi possível desenvolver website para gerenciar um catálogo de eventos da Comunidade Católica Colo de Deus e Santíssima Virgem, que oferece uma experiência agradável e intuitiva para os membros. ReactJS, NodeJS e MongoDB trazem inúmeros benefícios na criação de um site de eventos dinâmico e atrativo para o uso interno da comunidade, sendo assim, quando incorporada pelo cliente em seu domínio principal, gerará facilidade na comunicação, divulgação de informações e interação entre os participantes em escala local e global.

Palavras-chave: Website. React. Sites Modernos. Node. Mongo. Comunidade Colo de Deus.

ABSTRACT

Certainly, a website is an essential tool in the digital era and can be created using various technologies available in the market, such as React, a popular and efficient JavaScript library for building user interfaces. With this technology, it was possible to develop a website to manage an events catalog for the Colo de Deus and Santíssima Virgem Catholic Community, providing a pleasant and intuitive experience for its members. ReactJS, NodeJS, and MongoDB bring numerous benefits in creating a dynamic and appealing events website for internal community use. Therefore, when incorporated by the client into their main domain, it will facilitate communication, information dissemination, and interaction among participants on a local and global scale.

Keywords: Website. React. Modern Sites. Node. Mongo. Colo de Deus Community.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

UX	Experiência do usuário
CRUD	*Create*, *Read*, *Update and Delete* (Criar, Ler, Alterar e Deletar)
SGBD	Sistema de Gerenciamento de Banco de Dados
JSON	*JavaScript Object Notation* (Notação de Objeto JavaScript)
BSON	*Binary* JSON (Notação Binária de Objeto JavaScript)
API	*Application Programming Interface* (Interface de Programação de Aplicação)
NPM	*Node Package Manager* (Gerenciador de Pacotes Node)
E/S	Entrada e Saída
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
ID	*Identity* (identidade)
SPA	*Single Page Application* (Aplicação de Página Única)
DOM	*Document Object Model* (Modelo de Objeto de Documentos)
CSS	*Cascading Style Sheets* (Folhas de Estilo em Cascata)
req	*Request* (Requisição)
res	*Response* (Resposta)

SUMÁRIO

1 INTRODUÇÃO

A presença online, materializada por meio de um website, representa uma ferramenta incontornável para a comunicação eficaz em uma era digital (Czerniewicz et al., 2019). Em especial, para comunidades religiosas, como a Comunidade Colo de Deus e Santíssima Virgem, um website assume um papel crucial, oferecendo uma plataforma eficiente para a comunicação com os membros, disseminação de informações cruciais, promoção de eventos e angariação de fundos. A comunidade atende pelo site: colodedeus.com e o projeto integrará nesse domínio.

O Concílio Vaticano II, no âmbito da Igreja Católica, sublinha a importância da adaptação às transformações sociais e culturais contemporâneas, reconhecendo os meios de comunicação como instrumentos legítimos para a disseminação da mensagem religiosa (Vatican II, 1963). Nessa perspectiva, a iniciativa de desenvolver um sistema fundamentado em tecnologias como MongoDB, NodeJS e ReactJS está alinhada com a busca pela eficácia na comunicação, possibilitando a gestão independente e não relacional de eventos.

A criação de uma base de dados para eventos, com funcionalidades de cadastro, edição, atualização e exclusão, respalda-se na abordagem NoSQL do MongoDB, que confere flexibilidade e escalabilidade (Chodorow, 2013). Adicionalmente, a escolha de tecnologias como NodeJS e ReactJS coaduna-se com a busca por eficiência no desenvolvimento ágil de software, seguindo os princípios delineados pelo Manifesto Ágil (Beck et al., 2001).

Ao explorar a interseção entre a presença online, as orientações do Concílio Vaticano II e as melhores práticas em desenvolvimento de software, esta pesquisa visa oferecer uma base teórica sólida para a compreensão do papel crucial do website na comunicação eficaz de comunidades religiosas.

2 COMUNIDADE COLO DE DEUS E SANTÍSSIMA VIRGEM

Durante o Concílio Vaticano II, realizado entre 1962 e 1965, um marco na história da Igreja Católica, foi redefinindo muitos aspectos da vida e missão da igreja. Foi neste contexto que a compreensão do carisma, um dom especial do Espírito Santo, ganhou destaque. O carisma é uma força espiritual única que, quando discernida e vivida, contribui para a vitalidade e diversidade da comunidade cristã.

A Comunidade Colo de Deus e Santíssima Virgem, fundada há 19 anos (2004), representa um testemunho contemporâneo da atuação do carisma na vida da Igreja Católica. Seu surgimento remonta à visão e inspiração de Hugo Santos, junto, na época, por sua cônjuge Rosina da Cruz. A comunidade foi estabelecida no Rio de Janeiro, porém sua sede já passou por Curitiba- PR e Jandaia do Sul - PR, agora residindo em Maringá - PR, sendo presidida por Guilherme Martins, devido a renúncia de seu fundador.

A Colo de Deus emergiu como uma resposta ao chamado do Espírito Santo, promovendo uma experiência católica autêntica e envolvente. Segundo o estatuto da comunidade (Estatuto Colo de Deus, 2022), sua missão abrange diversos aspectos, desde atividades litúrgicas e formativas até iniciativas sociais, sendo uma comunidade Mariana, Pentecostal e Conciliatória, vivendo de acordo com a Doutrina Católica e fundamentados na espiritualidade da misericórdia. Refletindo assim a riqueza do carisma que a inspira, tendo o objetivo de trazer de volta para o seio da igreja católica aqueles que estavam afastados da fé, sendo instrumento de um grande avivamento. Logo, como é dito também no estatuto, pretende-se, então, como Colo de Deus ser, na igreja, esposos e esposas no Espírito Santo, gerando Cristo ressuscitado para o mundo.

Um dos eventos marcantes na história da Comunidade Colo de Deus é o evento Geração Atomika. Este encontro representa um momento de intensa espiritualidade e comunhão, unindo membros da comunidade em uma experiência única de fé e renovação. Este encontro anual, fortemente enraizado no carisma que guia a comunidade, é uma oportunidade para os membros se aprofundarem em sua fé, participarem de momentos de oração intensa e desfrutarem de atividades formativas. Sendo um evento aberto ao público, permite que novas pessoas conheçam o Carisma Colo de Deus e sejam convertidas ao catolicismo

Outro evento de destaque é a Conferência Yeshiva de Líderes. Esta conferência exemplifica o compromisso da Comunidade Colo de Deus com a formação e capacitação de seus líderes de Células, grupos de oração, normalmente realizados dentro das casas dos próprios líderes, promovendo partilhas e aprendizados espirituais.

A palavra "Yeshiva" carrega a conotação de um local de aprendizado profundo, e esta conferência reflete esse comprometimento com a educação e a liderança espiritual, levando o carisma às Nações, como uma confirmação ao ide, de Jesus Cristo, para pregar o evangelho pelo mundo.

Durante a Conferência Yeshiva de Líderes, líderes da comunidade se reúnem para aprofundar seu entendimento do carisma, explorar práticas pastorais eficazes e compartilhar suas experiências. Este evento é uma manifestação da busca contínua de excelência na liderança, mas também de imersão para os liderados, que também podem participar do evento, entendendo melhor como todo o processo é estabelecido para formação de uma Célula e sua necessidade na evangelização.

3 OBJETIVO

Num cenário contemporâneo, a presença online torna-se vital e, para as comunidades católicas, representa uma ferramenta inestimável para se conectarem com seus membros e com o mundo em geral. Um website não apenas proporciona uma comunicação eficaz com os fiéis, mas também viabiliza a divulgação de informações e eventos, a captação de recursos, a atração de novos membros e a educação sobre a religião.

Através da utilização de recursos online e plataformas digitais, a Igreja, representada pela Comunidade Colo de Deus, pode atingir um público mais amplo e amplificar sua voz no mundo moderno, compartilhando os ensinamentos e valores da Igreja Católica através do Carisma. Essa iniciativa é particularmente significativa para alcançar aqueles que, de outra forma, não teriam acesso a essas informações.

A acessibilidade emerge como um ponto crucial, considerando que a comunidade possui uma presença missionária não apenas no Brasil, mas em diversas partes do mundo, como Portugal, Itália, Reino Unido e Estados Unidos. Um website proporciona uma plataforma disponível 24 horas por dia, permitindo que católicos vinculados à comunidade ou simpatizantes do carisma permaneçam conectados, informados e engajados, independentemente de sua localização ou fuso horário.

Diante dessa necessidade premente, foi desenvolvido um sistema capaz de gerenciar eventos, permitindo a criação, edição e exclusão de informações referentes às atividades impulsionadas pela comunidade. O intuito é oferecer clareza e transparência nos dados apresentados ao usuário, consolidando-se como uma parte integrante de um corpo maior, ajustado para atender às demandas específicas do cliente.

Essa autonomia na gestão de eventos também visa contornar algumas limitações de plataformas terceirizadas, como o doity.com, reconhecendo a importância de um sistema próprio. Ainda que plataformas como o doity.com sejam populares, é necessário considerar fatores como taxas aplicadas por esses serviços. De acordo com informações disponíveis, o doity.com cobra taxas de 10% do valor do ingresso para eventos online, impactando diretamente nos custos finais e restringindo a margem de retorno financeiro para a comunidade (Doity, 2023).

4 ESPECIFICAÇÃO

4.1 EXPERIÊNCIA DO USUÁRIO (UX)

O sistema de eventos em questão é moldado para proporcionar uma experiência confortável para o usuário, através de uma aplicação, *one-single application,* uma abordagem de desenvolvimento de software na qual uma única aplicação é responsável por fornecer todas as funcionalidades necessárias. Nesta destacam-se elementos cruciais:

1. **Navegação Intuitiva:** Interface projetada para facilitar a exploração, garantindo uma interação natural e sem complicações;
2. **Feedback Imediato:** Respostas instantâneas a ações do usuário, criando uma experiência dinâmica e envolvente;
3. **Acessibilidade Prioritária:** Design que considera a acessibilidade, assegurando que todos os membros da comunidade possam participar plenamente;
4. **Consolidada Eficiência:** Oferece eficiência consolidada ao unificar todas as funcionalidades essenciais em uma única aplicação, simplificando a interação do usuário; e
5. **Desempenho Otimizado:** Minimiza a sobrecarga ao fornecer uma experiência de usuário contínua e otimizada, sem transições desnecessárias entre diferentes aplicativos.

4.2 ARQUITETURA DE SOFTWARE

A solidez da arquitetura é garantida pela escolha estratégica de tecnologias, incluindo MongoDB, ReactJS e NodeJS:

1. **Microsserviços:** Arquitetura baseada em microsserviços para manutenção eficiente e desenvolvimento independente de componentes;
2. **Eficiência com NodeJS:** *back-end* eficiente com NodeJS, proporcionando desempenho ágil e gerenciamento eficaz de conexões simultâneas;
3. **Flexibilidade MongoDB:** Utilização do MongoDB como banco NoSQL para adaptações rápidas a dados dinâmicos de eventos; e

4. **Integração Contínua:** Arquitetura suporta integração contínua, facilitando implementações ágeis e consistentes de melhorias.

4.3 PROTOTIPAGEM

A fase de prototipagem, realizada na plataforma Figma, foi fundamental para validar e aprimorar o sistema:

1. **Iteração Contínua:** Protótipos interativos permitiram iterações ágeis, refinando aspectos visuais e funcionais com feedback instantâneo; e
2. **Validação Antecipada:** Prototipagem permitiu validar conceitos precocemente, reduzindo riscos e garantindo alinhamento com as expectativas do cliente.

5 IMPLEMENTAÇÃO

A implementação de um sistema CRUD (*Create, Read, Update and Delete*) para um site de eventos, empregando MongoDB, Node.js e React.js, representa uma abordagem moderna e eficiente para o gerenciamento de informações. A combinação dessas tecnologias proporciona escalabilidade, eficiência na lógica de aplicação e interfaces interativas, culminando em uma solução completa e robusta para atender às necessidades de uma comunidade específica na organização de eventos.

5.1 SISTEMAS CRUD

CRUD é um acrônimo usado na área de tecnologia da informação e representa as quatro operações básicas do ciclo de vida de um dado em um sistema de gerenciamento de banco de dados. Essas operações são:

- ***Create* (Criação):** que permite a inserção de novos registros em uma tabela do banco de dados;
- ***Read* (Leitura):** que permite a consulta e recuperação de dados de uma tabela do banco de dados;
- ***Update* (Atualização):** que permite a atualização dos valores de um ou mais registros já existentes em uma tabela do banco de dados; e
- ***Delete* (Exclusão):** que permite a remoção de um ou mais registros de uma tabela do banco de dados.

O CRUD é uma das operações fundamentais em qualquer sistema de gerenciamento de banco de dados e é amplamente utilizado em sistemas e aplicativos que necessitam realizar operações de criação, leitura, atualização e exclusão de dados. É importante destacar que as operações do CRUD são as bases para implementação de funcionalidades mais complexas, como a validação de dados, buscas mais elaboradas e ordenação de resultados.

5.2 BANCO DE DADOS NÃO RELACIONAL: MONGODB

O MongoDB é um Sistema de Gerenciamento de Banco de Dados (SGBD) de código aberto e orientado a documentos. Ele foi desenvolvido para armazenar dados em documentos semelhantes a JSON (*JavaScript Object Notation*), permitindo uma maior flexibilidade na organização e estruturação dos dados, e por esse motivo o cliente optou por essa escolha para o catálogo de eventos.

Ao contrário dos bancos de dados relacionais tradicionais, que armazenam dados em tabelas com esquemas predefinidos, o MongoDB não possui esquema fixo, o que significa que você pode adicionar novos campos e coleções sem precisar definir a estrutura de antemão. Isso torna o MongoDB muito escalável e adaptável para diferentes tipos de aplicativos.

O MongoDB é amplamente utilizado em aplicativos web modernos, onde a escalabilidade e a flexibilidade são necessárias, especialmente em ambientes de nuvem e em sistemas distribuídos. Ele também possui uma linguagem de consulta poderosa que permite recuperar e manipular dados de maneira eficiente. O MongoDB opera bem com Node.js por algumas razões:

- **Ambos são baseados em JavaScript**: Tanto o MongoDB quanto o Node.js são baseados em JavaScript, o que significa que eles usam a mesma linguagem, o que torna mais fácil para os desenvolvedores integrá-los em seus projetos e manter um ambiente de desenvolvimento coeso;

- **Não há necessidade de transformação de dados:** O MongoDB armazena dados em formato JSON-like (BSON - *Binary* JSON), que é muito semelhante ao formato de objetos JavaScript. Isso significa que os dados podem ser enviados diretamente do MongoDB para o Node.js e vice-versa sem a necessidade de transformá-los em um formato diferente;

- **Operações assíncronas:** Tanto o MongoDB quanto o Node.js são projetados para trabalhar com operações assíncronas, o que significa que o banco de dados pode lidar com várias solicitações simultâneas sem bloquear o *thread* principal do Node.js. Isso torna o MongoDB especialmente adequado para aplicativos de alto desempenho que

precisam lidar com grande volume de solicitações; e

- **Compatibilidade com a biblioteca nativa do MongoDB para Node.js:** O MongoDB possui uma biblioteca nativa para Node.js chamada "*MongoDB Driver for Node.j*s", que oferece uma API (*Application Programming Interface* - Interface de Programação de Aplicação) completa e poderosa para interagir com o banco de dados. Essa biblioteca torna a integração entre o MongoDB e o Node.js mais fácil e eficiente.

Em suma, a combinação do MongoDB e Node.js é uma escolha popular para o desenvolvimento de aplicativos modernos, especialmente em projetos de alto desempenho e escalabilidade, devido à sua compatibilidade, eficiência e suporte para operações assíncronas. A escolha de um banco de dados não relacional, como o MongoDB, para realizar uma aplicação CRUD de elementos únicos, pode ser benéfica por várias razões:

- **Flexibilidade no modelo de dados**: O MongoDB é um banco de dados orientado a documentos que não possui um esquema de dados rígido, permitindo que os desenvolvedores armazenem e gerenciem dados de forma mais flexível. Isso significa que é fácil adicionar novos campos ou atualizar o modelo de dados sem precisar alterar a estrutura da tabela;

- **Escalabilidade:** O MongoDB é projetado para ser escalável horizontalmente, o que significa que você pode adicionar novos servidores e distribuir a carga de trabalho de forma mais eficiente do que nos bancos de dados relacionais. Isso pode ser útil quando você precisa lidar com grandes quantidades de dados ou quando espera um grande número de solicitações;

- **Desempenho**: O MongoDB é projetado para lidar com grande volume de solicitações e operações assíncronas, tornando-o uma boa opção para aplicativos que precisam lidar com um grande volume de leituras e gravações simultâneas;

- **Fácil integração com aplicativos modernos:** O MongoDB possui uma API simples e fácil de usar, além de ser compatível com uma ampla variedade de linguagens de programação, incluindo JavaScript (usado no Node.js), Python, Ruby e outras; e

- **Redução de complexidade:** Em comparação com os bancos de dados relacionais tradicionais, o MongoDB pode reduzir a complexidade do desenvolvimento de aplicativos, pois não requer um modelo de dados rigoroso ou complexas relações entre tabelas.

A estrutura do Mongo foi modulada através do seguinte código:

```
1  const mongoose = require ('mongoose');
2
3  module.exports = mongoose.model('Event', {
4      name: { type: String },
5      location: { type: String },
6      startedDate: { type: Date },
7      endedDate: { type: Date },
8      concluded: { type: Boolean },
9      banner: { type: String }
10 }, 'cdd');
```

Após o *download* do pacote mongoose no projeto, podemos importar o mesmo, indicado na linha 1 do código, e assim criar uma *Model* (estrutura de dados de uma entidade) para o documento do Mongo, indicados nas linhas 3 a 10 do código.

Dentro da estrutura podemos escolher o tipo de cada campo criado para o documento, indicados nas linhas 4 a 9, respectivamente, assim sendo:

- *name* (nome), do tipo *String* (cadeia de caracteres de texto);
- *location* (localização), também uma *String*;
- *startedDate* (data de início), do tipo *Date* (data);
- *endedDate* (data de término), também do tipo *Date*;
- *concluded* (concluído), do tipo *Boolean* (booleano - verdadeiro ou falso); e
- *banner* (estandarte). do tipo *String*.

5.3 BACK-END

O *back-end* de um sistema pode ser comparado ao mecanismo interno de um relógio, onde as engrenagens trabalham incansavelmente para manter o tempo correto, o *back-end*, muitas vezes imperceptível, é a força motriz que sustenta a operação contínua e eficiente do sistema como um todo. E no caso desse projeto, é impulsionado por tecnologias como Node.js, Express, e Yarn. Assim como as engrenagens, o Node.js fornece a base, enquanto o Express atua como a estrutura organizacional que direciona o fluxo de dados. Yarn, por sua vez, age como um eficiente gerenciador de dependências, garantindo a harmonia e eficácia na execução do sistema, tornando o *back-end* robusto e eficiente, como mecanismo complexo e preciso de um relógio bem projetado.

5.3.1 NODE.JS

Node.js é um ambiente de execução de código aberto que permite a criação de aplicativos de servidor utilizando a linguagem de programação JavaScript. Ele utiliza o mecanismo de execução V8 do Google Chrome para executar o JavaScript de forma eficiente. O Node.js foi projetado com base em uma arquitetura orientada a eventos e operações de entrada e saída não bloqueantes, o que o torna altamente escalável e capaz de lidar com um grande número de conexões simultâneas sem comprometer o desempenho.

Uma das principais vantagens do Node.js é a utilização do JavaScript como linguagem principal. O JavaScript é uma linguagem amplamente adotada e com uma grande base de desenvolvedores, o que facilita a transição e compartilhamento de código entre o lado do cliente e do servidor.

O NPM (*Node Package Manager*) é o gerenciador de pacotes padrão do Node.js, no entanto, o projeto utilizou o Yarn ao invés do nativo, isto porque o Yarn, em comparação ao NPM, destaca-se como uma escolha preferencial no gerenciamento de dependências para projetos JavaScript. Sua arquitetura eficiente e a capacidade de realizar instalações mais rápidas e consistentes conferem ao Yarn uma vantagem notável. Além disso, a capacidade de operar offline, o armazenamento de pacotes de forma mais eficiente e a resolução

determinística de dependências contribuem para tornar o Yarn uma opção superior, oferecendo uma experiência de desenvolvimento mais ágil e confiável.

O Node.js é frequentemente utilizado para o desenvolvimento de aplicativos em tempo real, devido à sua capacidade de lidar com eventos assíncronos e operações de entrada e saída de forma eficiente. Essa característica o torna adequado para cenários que exigem atualizações em tempo real, como bate-papo, *streaming* de dados e aplicativos colaborativos.

Em resumo, o Node.js é um ambiente de execução de código aberto que permite o desenvolvimento de aplicativos de servidor eficientes e escaláveis utilizando a linguagem JavaScript. Sua arquitetura orientada a eventos, ampla adoção do JavaScript e ecossistema robusto são pontos fortes que contribuem para sua popularidade e uso generalizado na comunidade de desenvolvimento.

Justificando a escolha, a combinação do ambiente de execução Node.js com o banco de dados MongoDB é uma escolha vantajosa para o desenvolvimento de aplicativos modernos. Node.js, baseado em JavaScript, e MongoDB, um banco de dados NoSQL orientado a documentos, possuem características que se complementam e oferecem benefícios significativos.

Primeiramente, a compatibilidade de linguagem entre Node.js e MongoDB facilita a integração entre o servidor e o banco de dados. Ambas as tecnologias utilizam JavaScript, o que permite o compartilhamento de código e simplifica o desenvolvimento e a manutenção do sistema.

Em relação à modelagem de dados, o MongoDB oferece flexibilidade significativa. Sua abordagem NoSQL permite que os desenvolvedores trabalhem com dados não estruturados ou sem uma estrutura de tabela fixa. Isso é especialmente útil em aplicações que requerem uma evolução ágil dos modelos de dados, já que permite adicionar, modificar ou remover campos sem a necessidade de esquemas rígidos.

Além disso, o Node.js é conhecido por sua capacidade de lidar com operações assíncronas e não bloqueantes. Isso é altamente compatível com as operações de entrada/saída (E/S) do MongoDB, permitindo que as consultas e operações com o banco de dados sejam

executadas de forma eficiente, sem bloquear a execução do código. Isso resulta em um melhor desempenho e escalabilidade do sistema.

No trecho de código abaixo, podemos ilustrar um exemplo de como opera um *controller* (controlador) no Node.js, por sua vez, controladores são uma parte da arquitetura de software responsável por lidar com as requisições HTTP, processar dados e interagir com o modelo de dados, garantindo a resposta apropriada para o cliente. Ele funciona como um intermediário entre as rotas (*endpoints*) da aplicação e a lógica de negócios.

```
1  router.post('/create', upload.single('banner'), (req, res, next) => {
2      const {name, location, startedDate, endedDate, concluded} = req.body;
3      if (!req.file) {
4            return res.status(400).json({ message: 'No file uploaded.' });
5      }
6      const bannerImagePath = req.file.path;
7      const event = {name, location, startedDate, endedDate, concluded,
banner: bannerImagePath};
8      eventCrud.create(event)
9            .then(data => res.status(201).json(data))
10           .catch(err => next(err));
11 });
```

Na primeira linha, declaramos o tipo de ação que desejamos realizar (neste caso, criar algo) com a requisição HTTP do tipo *POST*. Em seguida, especificamos o nome da rota e os parâmetros típicos (*req*, *res*, *next*), que são essenciais para o funcionamento da operação e são explicados mais à frente.

No restante do código, lidamos com os dados recebidos na requisição. Primeiro, verificamos se todos os campos necessários estão presentes (linha 2) e tomamos medidas para evitar possíveis problemas, como arquivos ausentes (linhas 3, 4 e 5).

Em seguida, construímos um objeto *event* com base nos dados recebidos do corpo da requisição (linhas 6 e 7). Este objeto contém informações importantes, como o caminho para o banner associado ao evento.

Por fim, utilizando uma função do *Model* chamada *eventCrud* e realizamos a criação do evento através da função *create* (criar) (linhas 7 a 10). Se a operação for bem-sucedida, responderemos com um código HTTP 201 (*Created*). Em caso de erro, emitimos uma mensagem de erro usando a função *next*.

5.3.2 MIDDLEWARES

Middlewares em Node.js são componentes ou funções intermediárias que atuam entre o servidor e as rotas de uma aplicação web. Eles desempenham um papel crucial no fluxo de requisições e respostas, permitindo implementar funções compartilhadas, processamento prévio ou pós-processamento, e controle do fluxo de execução. Os *middlewares* são inseridos entre a chegada da requisição ao servidor e o envio da resposta ao cliente, tendo acesso aos objetos de requisição e resposta. Eles seguem uma abordagem de "cadeia de responsabilidade", possibilitando modularizar a lógica de aplicação e executar tarefas específicas antes ou depois do processamento das rotas principais.

A biblioteca *Express*, já citada anteriormente, é um exemplo prático que permite criar *middlewares* personalizados para atender às necessidades específicas da aplicação, que por ventura foi a biblioteca que escolhemos para esse projeto. Falando de forma breve, o Express é um *framework* web minimalista para Node.js, projetado para facilitar o desenvolvimento de aplicações web e APIs de forma rápida e eficiente. Ele é uma das bibliotecas mais populares em Node.js e é amplamente utilizado pela comunidade de desenvolvedores devido à sua simplicidade, flexibilidade e robustez.

A seguir, exemplo de middleware utilizado no projeto para validar os ID 's no banco.

```
1  const validateDBID = (req, res, next) => {
2     if (!ObjectID.isValid(req.params.id)) {
3           res.status(400).json({
4                 error: 'O Id usado não é válido!',
5                 id: req.params.id
6           })
7     } else { next (); }
8  }
```

No trecho acima, definimos uma função middleware chamada validateBDID (linha 1), que utiliza os parâmetros típicos:

- **req:** Objeto de requisição (*request*), contendo informações sobre a solicitação do cliente;
- **res:** Objeto de resposta (*response*), usado para enviar respostas ao cliente; e
- ***next:*** Função que avança para o próximo middleware na cadeia de execução.

A partir da linha dois, realizamos verificações para garantir a validade do ID fornecido como parâmetro na requisição. Se o ID não for válido, o servidor responde com um código HTTP 400 (*Bad request*) em formato JSON (*JavaScript Object Notation*). Essa resposta inclui uma mensagem de erro "O ID usado não é válido!" (linha 4) e o ID fornecido pelo cliente (linha 5).

Assim, sempre que uma chamada à API contiver um ID como parâmetro, este middleware interrompe o processo, exibindo o erro. Caso a validação seja bem-sucedida, o middleware aciona a função *next* (linha 7), permitindo a continuidade na execução.

5.4 FRONT-END

O *front-end* de um sistema é como o mostrador elegante de um relógio, onde ponteiros meticulosamente projetados movem-se graciosamente para indicar o tempo. Assim como o design refinado de um mostrador, o *front-end* é a interface que os usuários vêem e interagem. Neste projeto, React atua como o arquiteto habilidoso que dá vida a essa interface, enquanto Axios funciona como um mensageiro ágil, facilitando a comunicação eficiente entre o *front-end* e o *back-end*.

Cada elemento no *front-end*, desde os ponteiros que apontam para a informação até a haste da pulseira, é cuidadosamente orquestrado para criar uma experiência de usuário fluida e envolvente. O Yarn, mencionado anteriormente, age como o maestro por trás dos bastidores, coordenando as dependências do projeto.

5.4.1 REACT.JS

React é uma biblioteca *JavaScript* de código aberto desenvolvida pelo Facebook, destinada a facilitar a criação de interfaces de usuário interativas e componentizadas em aplicações *web*. Desde o seu lançamento, o React tem se destacado como uma das ferramentas mais populares e amplamente adotadas pela comunidade de desenvolvedores, especialmente para o desenvolvimento de aplicações de página única (SPAs - *Single Page Applications*).

Uma das características fundamentais do React é o conceito de componentização (figura 1). Por meio desse paradigma, a interface do usuário é decomposta em componentes reutilizáveis e independentes. Essa abordagem modular permite a construção e a manutenção eficientes de aplicações complexas, ao mesmo tempo que promove uma organização lógica do código. Cada componente do React possui seu próprio estado (*state*) e propriedades (*props*) (figura 1), o que facilita o gerenciamento e a manipulação de dados na aplicação.

Outro aspecto relevante do React é a utilização do *Virtual* DOM (DOM Virtual). Esse mecanismo consiste em uma representação leve e abstraída da estrutura do DOM real. Ao realizar alterações na aplicação, o React compara o *Virtual* DOM com o DOM real e, com base nas discrepâncias encontradas, realiza atualizações mínimas e eficientes na interface de usuário. Dessa forma, o *Virtual* DOM otimiza o desempenho da aplicação, tornando-a mais ágil e responsiva.

A natureza reativa do React é essencial para proporcionar uma experiência de desenvolvimento mais intuitiva e responsiva. Quando o estado de um componente é alterado, o React automaticamente reage a essa mudança, atualizando a interface de usuário de acordo. Isso simplifica a lógica do desenvolvimento, uma vez que as mudanças no estado conduzem a atualizações automáticas na interface, reduzindo a necessidade de manipulações manuais e tediosas.

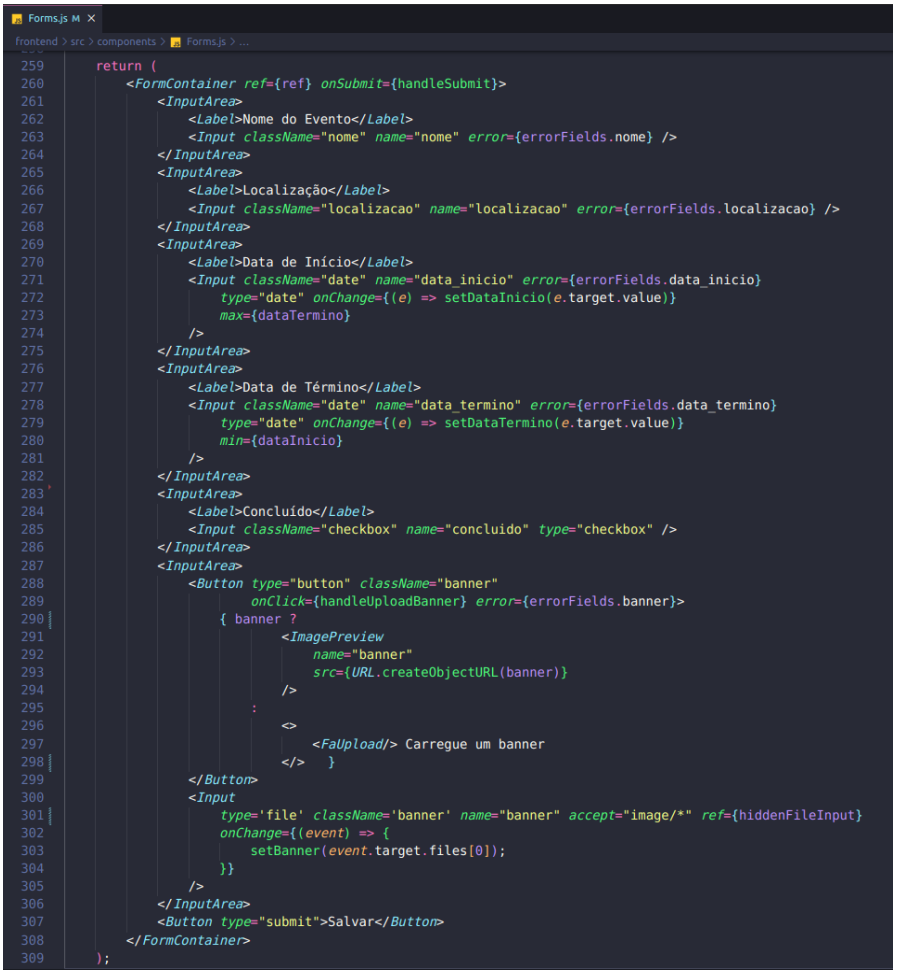
No contexto específico do projeto que envolve o gerenciamento de eventos em uma aplicação de página única, a adoção do React como tecnologia *front-end* é altamente recomendável. Com a habilidade de criar componentes reutilizáveis e independentes, o React permitiria a construção de uma interface de usuário dinâmica e flexível. Componentes específicos para a exibição e interação com eventos, como calendários, formulários e cartões de eventos, foram desenvolvidos e reutilizados ao longo da aplicação.

Ao combinar o React com tecnologias como Node.js no *back-end* e MongoDB como banco de dados, juntamente com o uso do framework Express para a construção da aplicação, seria possível criar um ambiente de desenvolvimento completo e integrado. O React agiria como a camada de visualização, facilitando a interação do usuário com os dados gerenciados pelo Node.js e armazenados no MongoDB.

Em resumo, o React é uma biblioteca de destaque no desenvolvimento de interfaces de usuário interativas e componentizadas. Sua abordagem modular, reatividade e integração com outras tecnologias tornam-no altamente recomendável para o desenvolvimento do projeto.

Na figura 1 abaixo, temos o componente Forms, com os componentes em azul claro, as *props* e algumas *functions* (funções) em verde, e os *states* em roxo. Note que as *props* sempre estão dentro do componente e as *functions* sempre acompanham parênteses "( )".

Figura 1 - Componentização no React, com *props*, *functions* e *states*



Fonte: Elaborado pelo autor (2023)

5.4.2 FIGMA

O Figma é uma ferramenta de design de interface de usuário baseada na web, que desempenha um papel significativo no desenvolvimento *front-end*. Ela se destaca por sua usabilidade intuitiva, recursos avançados e capacidade de colaboração.

Uma das principais razões para a importância do Figma no desenvolvimento *front-end* é sua capacidade de facilitar a colaboração entre designers e desenvolvedores. A ferramenta permite que várias pessoas trabalhem simultaneamente em um projeto, compartilhem feedback e visualizem alterações em tempo real. Isso aprimora a comunicação entre as equipes e agiliza o processo de design.

O Figma também oferece recursos avançados de prototipagem, permitindo que os designers criem protótipos interativos para demonstrar o fluxo de interação e a experiência do usuário. Esses protótipos podem ser compartilhados com a equipe de desenvolvimento para uma compreensão mais clara das interações planejadas e facilitar a implementação no *front-end*.

Outro benefício do Figma é a capacidade de criar componentes reutilizáveis e estilos globais. Isso promove a consistência visual em todo o projeto e simplifica a manutenção do design à medida que o projeto evolui. Os componentes podem ser facilmente atualizados e sincronizados em várias telas, reduzindo a duplicação de código e acelerando o processo de implementação.

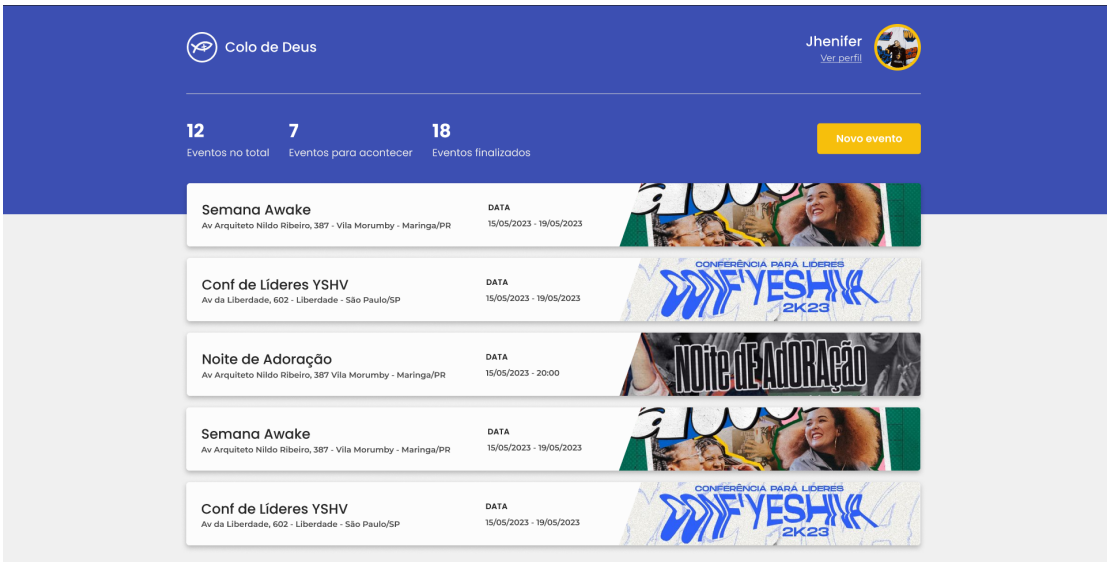
Além disso, o Figma oferece recursos de inspeção e exportação de elementos de design, como cores, tamanhos e espaçamentos. Essas informações são úteis para os desenvolvedores *front-end*, pois podem ser usadas para implementar o design com precisão. Além disso, algumas ferramentas e plugins do Figma permitem a geração automática de código CSS (*Cascading Style Sheets* - Folhas de estilo em cascata), agilizando ainda mais o processo de desenvolvimento *front-end*.

Utilizando dessa ferramenta foi confeccionado um *Wireframe* para esse projeto, ou seja, uma representação esquemática e simplificada de uma interface de usuário, usada para esboçar a estrutura e o *layout* de um projeto, sem se preocupar com detalhes visuais. Claramente uma ferramenta útil para a visualização e a comunicação das principais

características e funcionalidades da interface, permitindo uma abordagem interativa e colaborativa durante o processo de design.

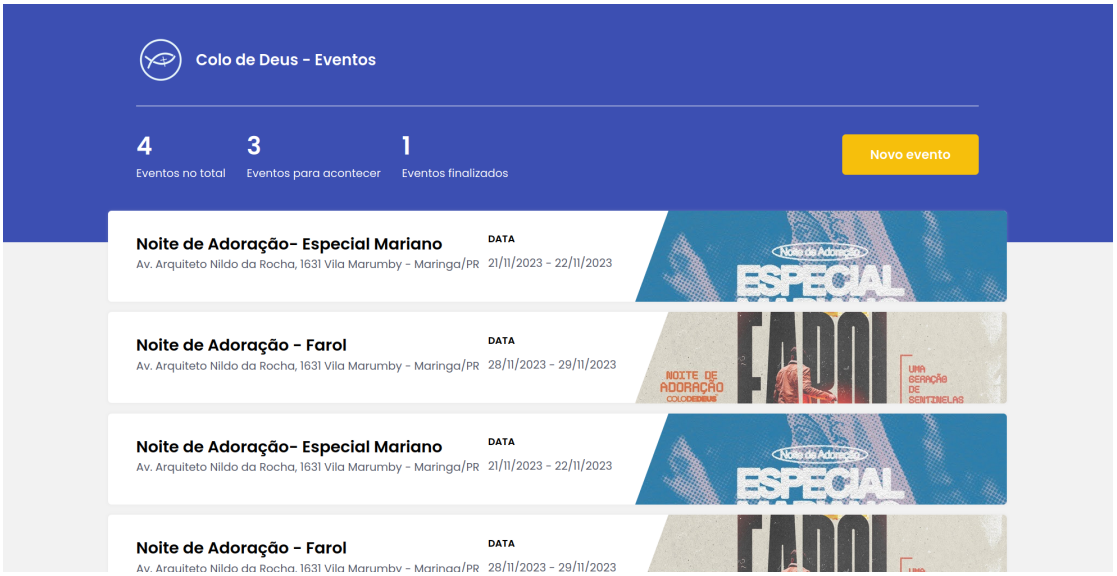
Segue na figura 2, o *wireframe* do projeto *one-single application* para comunidade Colo de Deus e Santíssima Virgem construído no Figma em comparação a versão final construída no React.js (figura 3).

Figura 2 - *Wireframe* do projeto Colo de Deus - Eventos



Fonte: Elaborado pelo autor (2023)

Figura 3 - Projeto Colo de Deus - Eventos em React.js



Fonte: Elaborado pelo autor (2023)

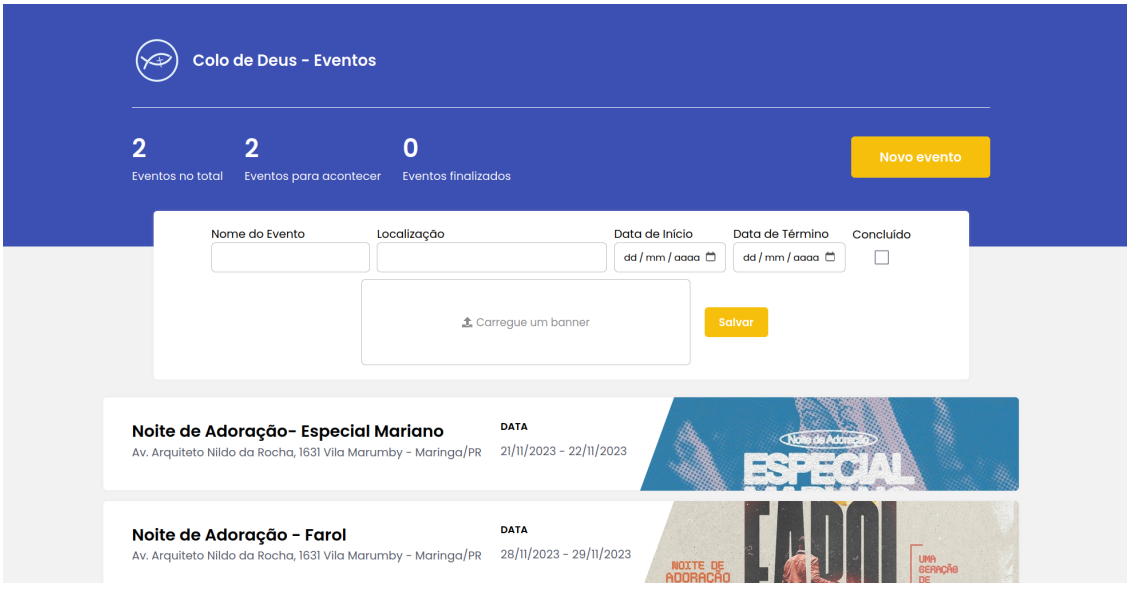
6 MODO DE USO

6.1 CADASTRO

Para cadastrar um novo evento o usuário deve clicar no botão “Novo evento”, que por sua vez irá abrir um formulário (figura 4) onde será possível inserir o Nome do Evento, o Endereço, a Data de Início e a Data de Término do mesmo, e realizar o *upload* de um *Banner*. Além disso existe a opção de ser um evento já encerrado, logo o usuário pode selecionar a caixa de concluído para registrar essa informação. Preenchendo todos os campos e clicando em “Salvar”, após a verificação interna e validação de dados, um aviso confirmando o cadastro é emitido e o evento é registrado (figura 5).

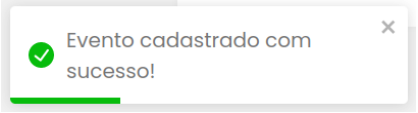
Em caso de algum dos campos obrigatórios (nome, localização, data de início, data de término e banner) não seja preenchido, um aviso será emitido ao usuário e o castrado não será realizado (figura 6).

Figura 4 - Formulário de cadastro



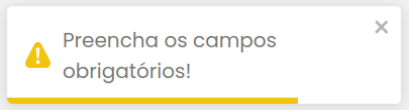
Fonte: Elaborado pelo autor (2023)

Figura 5 - Confirmação ao usuário sobre o sucesso no cadastro



Fonte: Elaborado pelo autor (2023)

Figura 6 - Aviso ao usuário sobre o preenchimento dos campos



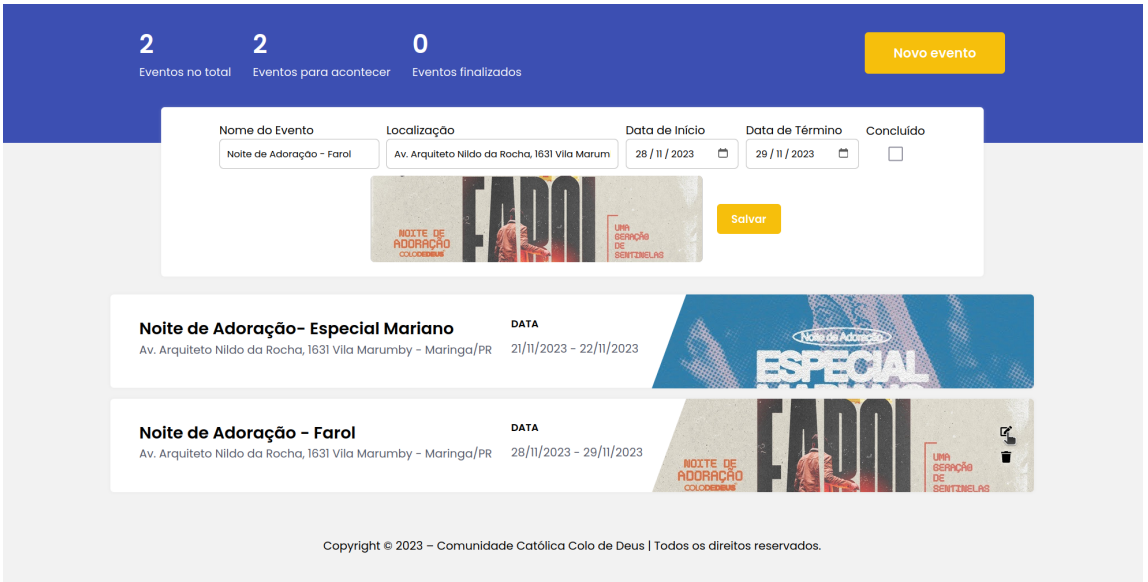
Fonte: Elaborado pelo autor (2023)

6.2 EDIÇÃO

Para realizar qualquer tipo de alteração no evento, basta ir até o evento que deseja editar, aproximar o mouse ao canto direito correspondente ao evento e aparecerá o ícone de "lápis" para a edição. Clicando no mesmo, o formulário será preenchido com todos os dados correspondentes desse evento em questão (figura 7), basta escolher o que deseja mudar e clicar em "Salvar" para confirmar a alteração.

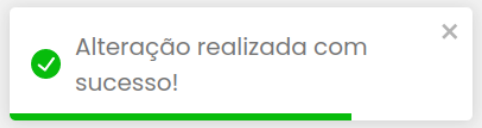
Após a verificação e validação interna, o usuário será notificado sobre o sucesso da sua operação (figura 8). Em caso do usuário excluir as informações e tentar salvar campos em branco, uma notificação de aviso será emitida (figura 6).

Figura 7 - Formulário após o usuário clicar no ícone de edição



Fonte: Elaborado pelo autor (2023)

Figura 8 - Confirmação ao usuário sobre a alteração realizada



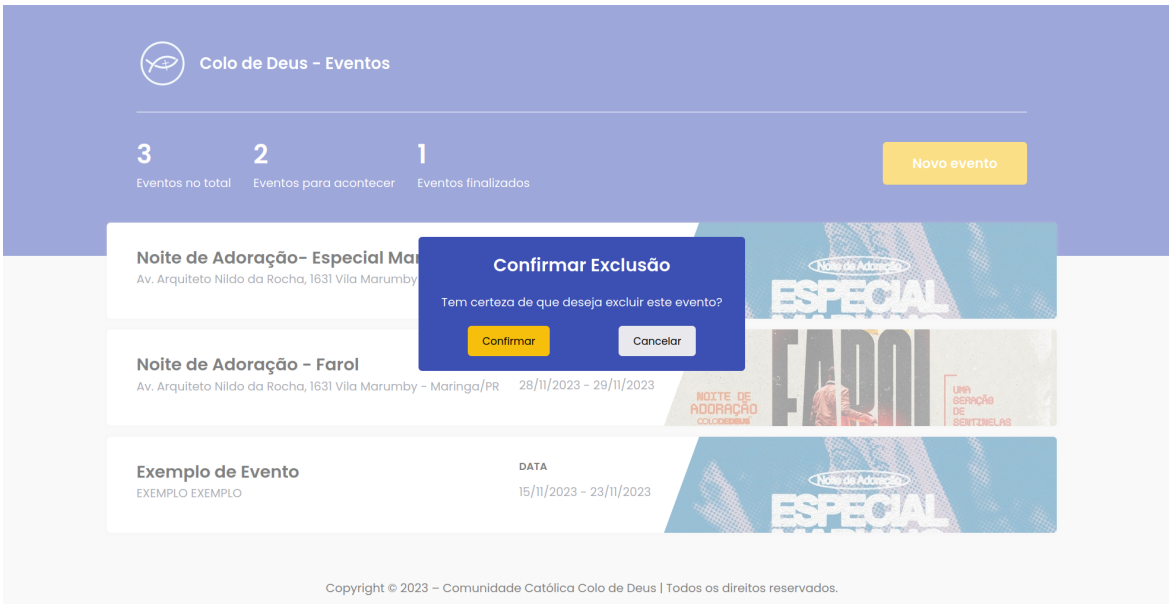
Fonte: Elaborado pelo autor (2023)

6.3 EXCLUSÃO

Quando o usuário desejar excluir algum evento, basta ir ao evento excludente aproximar o mouse ao canto direito e será apresentado o ícone de “lixeira” para exclusão, assim como na operação de edição. Clicando no ícone indicado, um aviso de operação de risco é exibido na tela (figura 9), pedindo a confirmação da exclusão, para que assim nenhuma operação acidental aconteça. Selecionando a opção de “Cancelar” o aviso é fechado e o processo encerrado.

Em caso de selecionar a opção “Confirmar”, o evento em questão é excluído do banco de dados, e uma notificação ao usuário é emitida (figura 10), confirmando que a exclusão foi realizada com sucesso.

Figura 9 - Aviso de operação de risco, confirmação de exclusão



Fonte: Elaborado pelo autor (2023)

Figura 10 - Confirmação ao usuário sobre o sucesso ao excluir

Fonte: Elaborado pelo autor (2023)

7 CONCLUSÕES

O desenvolvimento de um website de eventos para uma comunidade católica apresenta desafios e oportunidades únicas que podem ser abordados de forma eficiente com as tecnologias Node.js, MongoDB, Express e React. A escolha do Node.js para o *back-end* proporciona eficiência e escalabilidade, permitindo respostas rápidas às requisições dos usuários e gerenciamento eficiente do servidor. Além disso, a capacidade de trabalhar com JavaScript tanto no *front-end* quanto no *back-end* simplifica a colaboração entre as equipes de desenvolvimento.

O MongoDB, como banco de dados não relacional, é uma opção adequada para o projeto, permitindo o armazenamento ágil de dados relacionados aos eventos e membros da comunidade. Sua flexibilidade em relação à estrutura de dados e capacidade de escalabilidade horizontal garantem o tratamento eficiente de grandes volumes de informações.

A integração do Express ao projeto cria uma API robusta e segura para gerenciar as operações do servidor e manipular as requisições do cliente. O uso de middlewares personalizados facilita a implementação de tarefas específicas, como autenticação e controle de acesso.

Quanto à interface do usuário, o React é uma escolha adequada para criar uma experiência interativa e dinâmica. Sua abordagem de componentização permite a construção de elementos reutilizáveis para exibir e gerenciar os eventos da comunidade. O Virtual DOM do React garante atualizações rápidas e eficientes da interface, resultando em uma experiência de usuário fluida e responsiva.

O uso do Figma para design oferece a possibilidade de criar um wireframe detalhado e atraente da interface do website de eventos, facilitando o processo de desenvolvimento e a colaboração entre designers e desenvolvedores.

Diante da manipulação de arquivos utilizando o Multer, deparamo-nos com algumas dificuldades, como a duplicação de banners e desafios associados ao *upload* e exclusão de imagens nas extensões adequadas. Esses problemas indesejados foram identificados, superados e corrigidos ao longo do processo. A comunicação eficiente com a comunidade desempenhou um papel fundamental, permitindo que todas as tarefas fossem concluídas

conforme o cronograma estabelecido, mesmo diante das adversidades. Vale destacar que a questão relacionada aos arquivos, que se tornou um desafio, teve um prazo estipulado maior desde o planejamento, garantindo a conclusão bem-sucedida de todas as atividades.

Em suma, com as adversidades superadas, a combinação das tecnologias mencionadas fornece uma estrutura sólida para o desenvolvimento do website de eventos. Com eficiência, escalabilidade e flexibilidade, a aplicação oferece uma experiência positiva aos membros da comunidade católica Colo de Deus, sendo assim, quando incorporada pelo cliente gera facilidade na comunicação, divulgação de informações e interação entre os participantes em escala local e global.

REFERÊNCIAS

BECK, K.; BEEDLE, M.VAN BENDEGEM, A.; COCKBURN, A.; CUNNINGHAM, W.; ET AL. Manifesto for Agile Software Development, 2001.

CAMPBELL, Heidi. Digital Religion: Understanding Religious Practice in New Media Worlds. Nova York: Routledge, 2013.

COMUNIDADE CATÓLICA COLO DE DEUS E SANTÍSSIMA VIRGEM. *Estatuto.* Jandaia do Sul - PR / Brasil, 2022.

CONCÍLIO VATICANO II. *Atas do Concílio Ecumênico Vaticano II: Seções I a IV.* Petrópolis: Vozes, 1966.

CZERNIEWICZ, L.; DEACON, A.; GLOVER, M.; WALJI, S. A Wake-Up Call: Equity, Inequality and Covid-19 Emergency Remote Teaching and Learning, 2019.

DIGITALOCEAN. Tutorials - Pesquisa [MongoDB]. Disponível em: <https://www.digitalocean.com/community/tutorials?q=%5Bmongodb%5D&hits_per_page=12>. Acesso em: 15 de maio de 2023.

DOITY. Usar a Doity custa quanto?. Disponível em: <https://ajuda.doity.com.br/pt-br/article/usar-a-doity-custa-quanto-x7jqwv/>. Acesso em: 26 de Julho de 2023.

FIGMA. Figma. Disponível em: <https://www.figma.com/>. Acesso em: 17 de maio de 2023.

KAUFMANN, Jean-Claude. The Church and the Internet: Online Christian Communities. Nova York: Routledge, 2005.

MIKOWSKI, M. S.; POWELL, J. C. *Single Page Web Applications: JavaScript End-to-End*. Manning, 2013.

MONGODB. Documentação do MongoDB. Disponível em:<https://www.mongodb.com/docs/> Acesso em: 14 de maio de 2023.

MONGODB. Manual do MongoDB - Operações CRUD. Disponível em: <https://www.mongodb.com/docs/manual/crud/>. Acesso em: 15 de maio de 2023.

MONGODB INC. MongoDB: The Definitive Guide. 3rd ed. Sebastopol, CA: O'Reilly Media, 2019.

TUTORIALSPOINT. Node.js - RESTful API. Disponível em: <https://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm>. Acesso em: 16 de maio de 2023.