LUCAS FONSECA DOS SANTOS

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA GERAÇÃO DE CÓDIGO DE PÁGINAS WEB ESTÁTICAS:

DESAFIOS NO DESENVOLVIMENTO DE UM SISTEMA DE GERENCIAMENTO DE CONTEÚDO ESTÁTICO.

LAVRAS – MG

2022

LUCAS FONSECA DOS SANTOS

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA GERAÇÃO DE CÓDIGO DE PÁGINAS WEB ESTÁTICAS:
DESAFIOS NO DESENVOLVIMENTO DE UM SISTEMA DE GERENCIAMENTO DE CONTEÚDO ESTÁTICO.

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. DSc. Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG

2022

LUCAS FONSECA DOS SANTOS

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA GERAÇÃO DE CÓDIGO DE PÁGINAS WEB ESTÁTICAS: DESAFIOS NO DESENVOLVIMENTO DE UM SISTEMA DE GERENCIAMENTO DE CONTEÚDO ESTÁTICO.

DEVELOPMENT OF AN APPLICATION FOR GENERATION OF STATIC WEB PAGES CODE: CHALLENGES OF DEVELOPING A STATIC CONTENT MANAGEMENT SYSTEM

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 29 de Março de 2022.

Rafael Serapilha Durelli	DCC/UFLA
Maurício Ronny de Almeida Souza	DCC/UFLA
Lucas Fiorini Braga	Externo

Prof. DSc. Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG
2022

Dedico este trabalho a toda minha família que sempre tanto me apoiou e proporcionou, condições a alcançar essa etapa em minha vida. Ao meu pai, Luiz Carlos Baptista dos Santos, à minha mãe Carla Fonseca dos Santos e à minha irmã Fernanda Fonseca dos Santos, aos docentes que me transmitiram o conhecimento necessário e a comunidade que financia a educação pública de qualidade neste país.

AGRADECIMENTOS

Agradeço a Deus pela oportunidade de estar finalizando mais essa etapa em minha jornada. Agradeço ao meu professor e orientador Luiz Henrique Andrade Correia pela oportunidade na confecção deste projeto e por todo o auxilio prestado durante essa trajetória na Universidade Federal de Lavras, à minha família por todas as condições e apoio proporcionados até aqui, Luiz Carlos Baptista dos Santos, Carla Fonseca dos Santos e Fernanda Fonseca dos Santos e a todos os meus amigos que me acompanharam nessa jornada. Agradeço a todos os docentes que me lecionaram na Universidade Federal de Lavras e na escola onde estudei e formei minha base.

RESUMO

Em determinadas situações cujo o ambiente de infraestrutura é limitado, as capacidades operacionais e de performance de um software podem ser diretamente afetadas, surgindo a necessidade de recursos alternativos para sua operação efetiva. Neste contexto, no cenário do evento do XXIIX Congresso da Pós Graduação (CPG) da Universidade Federal de Lavras (UFLA) nasceu a ideia para o projeto discutido neste documento. Na ausência da possibilidade da infraestrutura disponível comportar um sistema robusto de *backend* e base de dados própria atrelado ao fator temporal do prazo, foi desenvolvido uma página web estática para apresentação do evento, populada via conteúdo armazenado em arquivos estáticos. Tendo em vista a problemática supracitada, foi construída a aplicação Open-Stacom, uma evolução do projeto inicial que trabalha com a geração de código de página estática para eventos. As páginas estáticas geradas podem ser hospedadas em uma infraestrutura sem a necessidade de base de dados e com baixo poder de processamento, reduzindo custos e complexidade de infraestrutura.

Palavras-chave: Arquitetura de Software; Aplicação Web; Geração de Código; Template; Frontend; Backend; Página Estática; Código Aberto;

ABSTRACT

In certain situations where the infrastructure environment is limited, the operacional and performance capabilities of a system can be directly affected, creating the need for alternative resources for its effective operation. In this context, in the XXIIX Congresso da Pós Graduação (CPG) event case of Universidade Federal de Lavras (UFLA) the project ideia discussed in this document was born. In the absence of possibility of available infrastructure to support a robust backend and its own database linked to the time factor of the deadline, a static webpage was developed to event presentation, populated from content stored at static files. In according to the aforementioned problem, the Open-Stacom application was built, an evolution of initial project that works with event static webpage code generation. The generated static webpages can be hosted in a simple infrastrucutre without the need of databases and with low processing power, reducing costs and the infrastructure complexity.

Keywords: Software Architecture; Web Application; Code Generation; Template; Frontend; Backend; Static Page; Open Source

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Este capitulo fará uma introdução do trabalho quanto a contextualização, motivação, objetivos e a metodologia.

1.1 Contextualização e Motivação

A decisão sobre quais modelos se implementar em um projeto talvez seja a das mais importantes decisões para se tomar no incio, dado que uma escolha assertiva de padrões arquiteturais, será decisivo para os pontos recorrentes citados e será definitivo ao todo o processo de desenvolvimento, em vista que uma mudança de arquitetura em meio ao progresso do processo produtivo de desenvolvimento poderá ser de enorme custeabilidade como tempo e custos financeiros, ou seja, a refatoração da estrutura do projeto, poderá implicar em prejuízos para empresas e consumidores.

Diferentes arquiteturas solucionam diferentes problemas em projetos de *software*, mas a adoção de um determinado modelo em detrimento de outro, pode resultar na variabilidade de certos fatores operacionais como um aumento ou redução da complexidade/eficiência dos principais pontos supracitados: escalabilidade, manutenção, evolução, segurança e desempenho, com isso, apresenta-se neste trabalho, uma análise dos resultados obtidos na implementação do projeto Open-Stacom, demonstrando o cenário imposto pelas condições da aplicação e requisitos bem como após a implementação de todo o código e os modelos.

O projeto do *software* Open-Stacom é um caso composto de diversas particularidades que permite uma boa análise e discussão para a compreensão das tomadas de decisão referente a qual modelo arquitetural se implementar. Para começar, a ferramenta Open-Stacom propõe a geração de *web* sites estáticos e de forma simples para o usuário final, páginas estáticas, pois a ideia nasce em um contexto de uma infraestrutura limitada disponível para hospedagem de um sistema completo de gerenciamento de conteúdo com prazo extremamente reduzido

para seu desenvolvimento, resultando na necessidade de se disponibilizar apenas uma página estática para apresentação das informações de um evento acadêmico, neste cenário, a página de apresentação do evento XII Congresso de Pós Graduação da Universidade Federal de Lavras (UFLA) ao invés de um sistema completo CMS (*Content Management System* ou Sistema de Gerenciamento de Conteúdo).

Objetivando uma melhoria nas condições de gerenciamento do conteúdo dessa página estática, nasce um projeto-piloto onde se disponibiliza todo o conteúdo da página via arquivos de transporte de dados em formato JSON (.json). A simplicidade e eficiência do manuseamento por parte do usuário final acarretou-se a uma boa aceitação da ferramenta e ao plano de sua evolução para a criação de uma *interface* de registro e atualização destes dados entre o usuário final e os arquivos responsáveis pelo armazenamento estático destes dados, disponibilizado através de um serviço *web*.

Nessa etapa, analises apontaram para uma deficiência das práticas SEO (*Search Engine Optimization*) ao se injetar de forma direta os dados que compõe os conteúdos nas *interfaces* das páginas *web*, após o momento de sua renderização, de forma assíncrona. Então, objetivando a solução deste problema, propôs-se a construção de uma aplicação mais robusta que injetasse essas informações anteriormente contidas em arquivos estáticos, direto no código-fonte das páginas *web* estáticas, na camada de *Front-end*.

Essa evolução no projeto trouxe diversos problemas de projeto em que foi necessário a adoção de modelos arquiteturais para à camada de *back-end*, responsável pela geração do código-fonte da página *template* estática e para a camada de *front-end* responsável pela entrada e visualização dos dados por parte do usuário final.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo do trabalho é apresentar uma discussão sobre a implementação de modelos arquiteturais no projeto da aplicação Open-Stacom, um gerenciador de conteúdo e gerador de página *web* estática baseado em *templates* pré definidos para *websites* de eventos acadêmicos e como essas arquiteturas foram utilizadas para sanar diversas questões recorrentes em um projeto de *software* como escalabilidade, manutenção e evolução, segurança e desempenho.

1.2.2 Objetivos Específicos

Orienta-se como objetivos deste trabalho, estudar o contexto e as necessidades da aplicação Open-Stacom, estudar e compreender diferentes modelos arquiteturais candidatos, efetuar um estudo comparativo entre tais modelos arquiteturais, definir um modelo arquitetural para a aplicação Open-Stacom e desenvolver um protótipo utilizando o projeto arquitetural definido.

A estrutura do trabalho está amparada em um formato onde o capítulo 2 contém o Referencial Teórico, apresentando o estado da arte e todos os principais conhecimentos utilizados no trabalho, o capítulo 3 apresenta os trabalhos relacionados cuja soluções apresentadas são similares a solução proposta apresentada neste trabalho, o capítulo 4 apresenta seções relacionadas ao desenvolvimento do projeto como Materiais e Métodos, o capítulo 5, detalha todo o problema de pesquisa, a solução para se alcançar o objetivo discutido e a aplicação desenvolvida com base no projeto arquitetural selecionado e o capítulo 6 explana as conclusões a partir dos resultados obtidos.

2 REFERENCIAL TEÓRICO

Este capitulo irá apresentar todo o referencial teórico utilizado para a construção do conhecimento no trabalho apresentado.

2.1 Arquitetura de Software

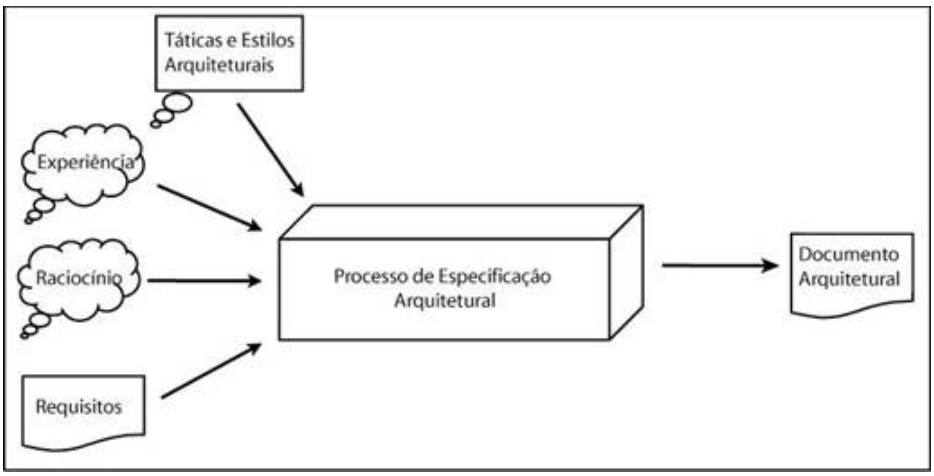
Segundo Martin 2019, modelos de arquitetura de *software* sempre objetivam a minimização de recursos humanos para o processo de construção e manutenção de um sistema.

Já Valente 2020, traz a definição de arquitetura como um modelo que visa sanar preocupações de alto nível no projeto de um sistema, abordagem essa representada por unidades de maior tamanho que representem na prática, um conjunto de classes ou *interfaces* como módulos, subsistemas, camadas ou serviços. Essa visão de mais alto nível busca uma garantia da escalabilidade e desacoplamento entre os diferentes componentes do sistema.

Um modelo arquitetural é ser construído com base em determinados insumos referentes ao contexto do projeto, que ao passar pelo processo de especificação, é gerado um documento de especificação arquitetural. Tal documento é constituído pela apresentação da organização de alto nível do projeto Figura: 2.1.

Ao longo dos anos, os avanços na área da tecnologia impuseram inúmeros novos desafios a arquitetura de *software* Garlan 2014 tornando o campo da arquitetura, uma promissora área de pesquisa, gerando assim, diversos novos modelos arquiteturais que objetivam sanar diferentes problemas no desenvolvimento de *software*, tornando assim, a decisão pela opção de qual modelo adotar em um projeto, uma das mais importantes decisões de projeto, visto que um modelo correto, auxiliará o sistema a atingir resultados satisfatórios em desempenho, portabilidade, escalabilidade, interoperabilidade e confiabilidade Garlan 2014.

Figura 2.1 – Elementos da construção de um projeto de arquitetura.



Fonte: https://www.devmedia.com.br/fundamentos-de-arquitetura-de-software/10581

Basicamente, pode-se dizer que o modelo arquitetural presente em um sistema é a ponte entre os requisitos funcionais e a implementação final do sistema conforme ilustrado na Figura 2.2

Figura 2.2 – Arquitetura de Software intermediando requisitos e implementação.

Fonte: Autor

2.1.1 Arquitetura Monolítica

2.2 *Search Engine Optimization* (SEO)

Com a popularização e a democratização do acesso à *internet* crescente, o principal meio de que um usuário da rede utiliza para encontrar qualquer categoria de informação que seja útil para sua rotina é por pesquisa em buscadores como Google, Yahoo, DuckDuckGO, dentre outros. Mais de 80% dos primeiros acessos de uma página disponibilizada na *internet* tem como origem alguma plataforma buscadora Zhu e Wu 2011.

Páginas disponibilizadas na *internet* são ranqueadas e listadas por motores de busca com base em quesitos específicos de conteúdo e estrutura, avaliados por algorítimos. As estruturas são avaliadas conforme boas práticas que regem a construção de páginas *web* (correto uso de etiquetas HTML — *HyperText Markup Language*). Já os conteúdos, são avaliados por meio dos termos textuais e relacionados as palavras chaves utilizadas pelo usuário na busca. Termos classificados de forma relevante dado a estrutura onde estão inseridos na página *web*, como, por exemplo, etiquetas H1 ou H2 do HTML são avaliados e relacionados com os termos chaves da busca ou textos contidos em etiquetas de *anchor link*. Quanto maior o grau de relacionamento, maior será a relevância para a página no ranqueamento executado pelo algorítimo, bem como melhor será a posição dessa página na listagem de resultados do buscador. No Google, por exemplo, um dos mecanismos de avaliação e ranqueamento é chamado *PageRank* ou GooglePR Page et al. 1999.

Segundo Zhu e Wu 2011, estatísticas demonstram que até 84% dos internautas não avançam além da segunda página e 65% dificilmente acessam resultados cujo posicionamento é pago ou patrocinado. Este fato, denota a importância de uma boa administração do conteúdo bem como uma boa construção da página seguindo corretamente os princípios que regem a estrutura HTML. O conteúdo deve estar apresentado da forma correta e os termos relevantes que o descrevem, na es-

trutura correta do HTML para que os algorítimos de ranqueamento posicionem a página entre os primeiros resultados.

Desenvolvedores e os administradores das páginas devem então considerar a implementação de um conjunto de regras descrito como *Search Engine Results Pages* (SERP) para posicionar suas páginas dentre os primeiros resultados listados pelos motores de busca, garantindo assim maior uma maior quantidade e assertividade dos acessos. Setiawan et al. 2020.

SERP é um sub conjunto de *Search Engine Optimization* (SEO), um conjunto de técnicas e boas práticas que permite aos desenvolvedores e administradores de páginas obterem um posicionamento satisfatório com maior relevância na indexação dos buscadores. Existe ainda outro sub conjunto denominado *Search Media Optimization* (SMO) que consiste em regras e procedimentos para melhores resultados em redes sociais.

A negligência dessas práticas atualmente pode resultar em um ocultamento da página e seu conteúdo frente a milhares de outros concorrentes, neste aspecto, haverá uma perda de competitividade. Já para aqueles que seguem os procedimentos previstos no SEO e implementam toda a estrutura e conteúdo planejado, poderá haver um aumento substancial de público visitante conforme demonstrado por Setiawan et al. 2020, aumentando a competitividade e exposição.

2.3 Usabilidade

A norma ISO 9240 – 11 Iso 1998 explana o termo usabilidade como um conceito que exprime a condução do usuário de um determinado produto (*software* para o presente caso) em atingir seu objetivo com toda a relação à satisfação, eficacia e eficiência. No que lhe concerne Nielsen 1994, expande essa definição de usabilidade adicionando o fator aprendizagem como "a capacidade de aprender e memorizar um sistema de *software*, a sua eficiência de uso, a sua capacidade de evitar e gerenciar erros de usuário e satisfação do usuário".

Angeli et al. 2003 complementa que, a usabilidade é um aspecto significante da qualidade global de sistemas interativos.

Com a expansão dos serviços digitais, um novo conceito entrou em cena, a experiência do usuário Soegaard e Dam 2012. A perspectiva da experiência do usuário se traduz no campo cognitivo humano, afetivo, social e físico da interação para além dos três fatores que compõe a usabilidade, de acordo com Martins et al. 2013.

Relacionada a usabilidade, a experiência do usuário uma vez negativa, pode acarretar uma não aceitação do produto no mercado, no fracasso, enquanto a boa experiência pode servir para impulsionar seu sucesso. Qualquer um dos três fatores presentes pertencentes a definição de usabilidade (satisfação, eficacia ou eficiência) na experiência do produto podem contribuir para uma experiência positiva ou negativa. Neste contexto, pouco se torna relevante qualquer investimento no projeto sem que estes fatores sejam alcançados, segundo Krug 2001.

Um projeto de *software* pode ser submetido a processos de evolução para que o fator usabilidade seja melhorado, existem diversas técnicas, por exemplo, demonstrado por Nielsen 1994 no livro *Usability Engineering* que determinam um conjunto de heurísticas que podem ser aplicadas a um sistema. Para este caso, diversos benefícios são apontados. Um nível mais alto de usabilidade, contribuirá para um maior desempenho de pessoas com necessidades especiais Martins et al. 2012; usuários com menor competência e familiaridade com tecnologia reduzindo a exclusão tecnológica para aqueles com menor aptidão técnica; uma melhoria da aceitação por parte dos usuários; uma redução da curva de aprendizagem para utilizar o produto; uma redução da probabilidade da ocorrência de erros do usuário; um aumento da produtividade advinda da qualidade de interação e um aumento dos fatores parte da usabilidade: satisfação, eficácia e eficiência, visto que os objetivos do usuário, são alcançados com maior facilidade Martins et al. 2012, Bevan 1998, Bevan, Claridge e Petrie 2005.

2.4 *Design* Responsivo

Segundo informações levantadas por Insights 2013, as pesquisas demonstram até julho de 2012 que 61% dos usuários cujo trafego tem como origem algum dispositivo mobile, ao se depararem com um site não totalmente adequado ao dispositivo, são mais propensos a visitarem e escolher um *website* concorrente. Também constatou-se que, para este período de 2012, cerca de 75% dos usuários que visitam uma determina página *web*, utilizam alguma categoria de dispositivo mobile como celular ou *tablet*. Ainda compilado por este mesmo estudo, um total de 74% dos visitantes de uma página *web* podem abandonar o acesso ou perder o interesse dado uma velocidade de carregamento da *interface* superior a 5 segundos.

De acordo com este levantamento, é possível afirmar que uma página *web* cujo *design* e a organização de sua estrutura não esteja adequada aos novos padrões mobile, poderá sofrer uma grande perda de interesse e engajamento, refletindo em uma rejeição de seu público alvo. Para se manter competitivo no mercado atual, faz-se necessária a adaptabilidade a qualquer dispositivo que permita acesso ao seu conteúdo.

Design responsivo é a capacidade de adaptação e apresentação do conteúdo e a estrutura de uma página *web* para diferentes dispositivos sob diferentes resoluções de visualização da *interface* gráfica. Com o enorme crescimento do uso de dispositivos mobiles como celulares e *tablets* além de telas de visualização em tamanhos variados presentes em dispositivos IoT (Internet of things — Internet das coisas), o desenvolvimento *web* com abordagem do *design* responsivo se tornou um desafio aos desenvolvedores Wongsalam e Senivongse 2019, que passaram a se preocupar com a implementação de regras de adaptação e apresentação para as diferentes resoluções existentes no mercado e para os *designers*, a responsabilidade do planejamento e prototipação de diferentes *layouts* de apresentação do conteúdo e readequação das estruturas de uma determinada *interface*.

As tecnologias acompanharam essa evolução e implementaram novos mecanismos que facilitam o trabalho dos desenvolvedores. A linguagem de estilização de páginas *web*, CSS evoluiu para uma nova versão nomeada como CSS3, com o surgimento das regras de *media queries* que definem diferentes formas de apresentação dado a condicional da resolução do dispositivo, orientação (paisagem, retrato) ou modo (impressão, visualização) Wiener, Ekholm e Haller 2015.

3 TRABALHOS RELACIONADOS

Neste capítulo, apresentam-se trabalhos que abstraem soluções similares à proposta discutida neste trabalho.

3.1 Sistema EasyChair

EasyChair é um sistema desenvolvido de plataforma *web* desenvolvido em 2002 por Andrei Voronkov, professor no Departamento de Ciência da Computação da Universidade de Manchester. Classificado como um sistema do tipo *Event management software*, é uma plataforma que permite o gerenciamento da informação para eventos de nicho acadêmico. O sistema comporta eventos de conferência *single-track* com um único comitê de programa e conferencias *multi-track* com vários programas e um comitê para cada. Lo, Phan e Goi 2007.

A plataforma gerencia a submissão e avaliação de trabalhos e documentos com toda a gestão de banca, revisores e *feedback*. Além disso, EasyChair possui um módulo de gestão de conteúdo para apresentação do evento em uma página *web* gerada, como um CMS (*Content Management System*) onde todas as informações públicas sobre o evento podem ser divulgadas aos interessados e essa página publicada em um serviço *web* juntamente ao sistema.

A aplicação possui licença proprietária, também possui versão gratuita e paga. Para eventos acadêmicos de pequeno e médio porte, as *features* disponibilizadas na versão gratuita conseguem atender bem necessidades e requisitos mais simples como as funcionalidades básicas de submissão e revisão de documentos, gerenciamento de conteúdo e apresentação do evento em página *web*, gerenciamento de programas da conferência e administração de contas de usuário, administradores e revisores, conforme tabela de *features* apresentadas no site oficial EasyChair Conference System.

Entretanto, problemas de outra natureza podem ser um fator limitante ao usuário. O sistema desenvolvido em 2002 possui uma usabilidade limitada com

um projeto de *interface* gráfica e *design* já defasados, o que pode impedir o eficiente e assertivo uso por parte de um usuário com menor competência tecnológica. O projeto da *interface* gráfica também não é responsivo (não se adapta a outras resoluções de tela e dispositivos) tornando assim, inviável seu uso em plataformas que não seja um computador com resolução superior a 1024px de largura por 768px de altura.

A página *web* gerada para apresentação e divulgação do evento na *internet* também não possui *interface* gráfica responsiva, reduzindo o alcance da divulgação do evento para o conjunto dos usuários cuja origem de acesso advém de algum dispositivo mobile.

3.2 HotCRP

A aplicação HotCRP é uma plataforma para revisão de submissões de trabalhos e documentos para eventos de conferência de nicho acadêmico. Com licença proprietária e possibilidade de uso gratuito somente para conferências patrocinadas pela ACM ou pela USENIX, possui um custo de $7.50 por submissão para eventos não associados, podendo assim tornar, inviável o uso para eventos de menor porte que não possua condições financeiras de aderir ao seu uso.

3.3 SCOOCS IChair

IChair é um projeto moderno que provém aos usuários a possibilidade da construção e realização de um evento de conferência totalmente *online*, com toda a estrutura necessária como salas virtuais, salas de *livestream*, agenda de atividades, e análise dos dados gerados ao decorrer do evento. Com *design* moderno e responsivo, o sistema se adéqua para dispositivos *mobile* e possui uma capacidade maior para apresentação dos dados em seu *dashboard* principal.

Entretanto, a plataforma de licença proprietária não possibilita seu uso de forma gratuita, podendo minar a capacidade de eventos de menor porte aderir ao

seu uso. Além disso, o produto não é focado na apresentação e divulgação do evento na *internet*, conforme seria o objetivo de um módulo CMS, por exemplo, necessário o desenvolvimento a parte de uma página web para suprir tal necessidade.

3.4 CMS Wordpress

Wordpress é uma plataforma CMS (*Content Management System* ou Sistema Gerenciador de Conteúdo) livre (GPLv2 — *General Public License*) e *open-source* mantido pela *WordPress Foundation* que trabalha com páginas *web* baseadas em *templates* (conjunto de padrões e estilo de páginas *web*). Desenvolvedores podem implementar novos *templates* a associá-los ao código do Wordpress para que sua estrutura *back-end* interaja com a camada *front-end* implementada.

Para administrar o conteúdo de um *website* Wordpress, a plataforma fornece um *dashboard* de gerenciamento onde o usuário administrador consegue efetuar suas operações de CRUD (*Create*, *Read*, *Update* e *Delete*) com seu conjunto de informações que deseja publicar no site. O produto também conta com uma enorme comunidade que desenvolve *plug-ins* e *scripts* que podem ser adicionados ao projeto para possibilitar novas funcionalidades ao usuário. É uma plataforma muito robusta, com um elevado grau de atualizações e delega questões como usabilidade e experiencia do usuário (sobre a ótica do usuário final, que visita o site) ao *template* implementado.

Para se publicar uma página desenvolvida com Wordpress, é necessário dispor de uma infraestrutura miníma com banco de dados e um servidor *web* PHP para instalar a plataforma *back-end*, em determinados casos, isso pode constar como um fator limitante. Não há custos empregados ao usuário final, é gratuito. Poderá haver custos em *templates* desenvolvidos e comercializados por terceiros.

3.5 CMS Joomla

A plataforma Joomla, classificada como um Sistema Gerenciador de Conteúdo, é uma das mais poderosas aplicações *Open-Source* deste tipo. Desenvolvido com objetivo de comportar várias categorias de projetos como *E-commerce*, portais e *blogs*, trabalha com um sistema de *templates* desenvolvidos por terceiros, comercializados e gratuitos Patel, Rathod e Parikh 2011. A aplicação fornece funcionalidades padrões de *back-end* interligadas ao *template* utilizado pelo usuário e ainda fornece diversos módulos opcionais denominados *Add-ons*.

A plataforma base oferece as funcionalidades de gerenciamento de usuários, conteúdo, contatos, *banners*, *template*, dentre outros.

Para disponibilizar e gerenciar uma página integrada ao Joomla na *internet*, faz-se necessário a disponibilidade de uma infraestrutura composta por um servidor *web* PHP com banco dados em MySQL (Apache) para a instalação da plataforma Dorosh e Kuchmij 2009.

4 MATERIAIS E MÉTODOS

Apresenta-se neste capítulo, de forma detalhada todas as técnicas e processos utilizados durante a construção deste trabalho.

No ano de 2019, foi desenvolvido um projeto de CMS (*Content Management System*) estático para administrar o conteúdo e divulgar o o evento XVIII Congresso da Pós-Graduação da Universidade Federal de Lavras. A página *web* com as informações do evento carregava as informações da conferência de forma estática, visto existir uma limitação de infraestrutura para comportar uma aplicação mais robusta com sistema de gerenciamento de conteúdo e banco de dados.

Visando uma melhoria de SEO (*Search Engine Optmization*) na página *web* gerada como saída do processamento dessa aplicação e um ganho significativo de usabilidade por parte dos usuários finais, foi elaborado uma evolução para este projeto, uma nova versão, onde se considerou um modelo arquitetural diferente bem como uma rotina de processamento também diferente explanado neste capítulo.

Como supracitado, o objetivo deste estudo envolve a construção de um sistema (Open-Stacom — *Open Source Static Content Management tool*) que possui como função principal a geração de páginas *web* estáticas para apresentação e divulgação de eventos acadêmicos do tipo conferencia.

Figura 4.1 – Logotipo do projeto Open-Stacom.

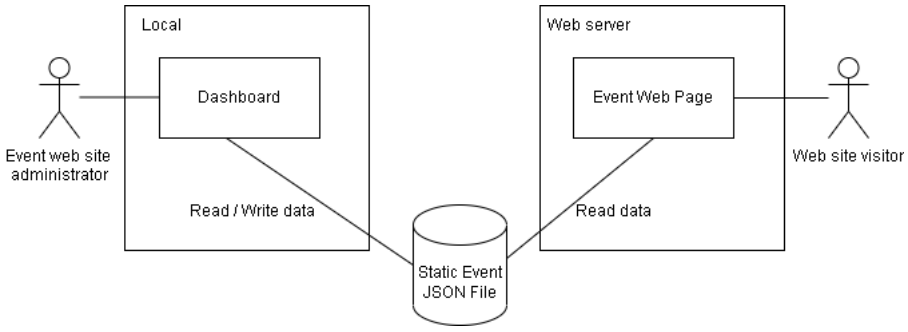
Fonte: https://github.com/LucasFonsecaDosSantos/Open-Stacom

Para se manter a assertividade dos requisitos funcionais solicitados, a qualidade do *software* e a redução de erros, foi adotado um procedimento de engenharia de *software* para o desenvolvimento do produto final bem como uma série de procedimentos minuciosos.

4.1 Projeto Piloto Open-Stacom

A primeira versão da aplicação consistia em uma simples *interface* gráfica *front-end* utilizada para se manipular com operações de CRUD (*Create*, *read*, *update* e *delete*) um arquivo que continha um objeto Javascript para transporte de dados no formato JSON. Todas as informações pertinentes a um evento de conferência eram armazenadas em um arquivo com objeto JSON e eram carregados de forma assíncrona (Ajax) por rotinas escritas em linguagem Javascript da página *web* de apresentação do evento conforme mecanismo ilustrado na Figura 4.1.

Figura 4.2 – Diagrama básico da aplicação Open-Stacom versão 1.0.



Fonte: Autor

```
function loadJSON(callback) {
    var xobj = new XMLHttpRequest();
    xobj.overrideMimeType("application/json");
    xobj.open('GET', 'content/index.json', true);
    xobj.onreadystatechange = function () {
        if (xobj.readyState == 4 && xobj.status == "200") {
            callback(xobj.responseText);
```

```
8           }
9       }
10      xobj.send(null);
11  }
```

Listing 4.1 – Requisição Ajax Javascript utilizada na página de apresentação do evento.

A injeção do conteúdo ocorria toda de forma assíncrona, após o aplicativo *Browser* já ter iniciado a renderização dos elementos estruturais da página. Essa abordagem apresenta um problema para o ranqueamento da página *web* nos motores de busca, visto que o conteúdo que deveria ser analisado pelos algorítimos, não está presente de forma concreta na página.

Todo o conteúdo era carregado dinamicamente dado que existiam rotinas de tradução para todos os textos presentes na *interface* como chamadas em botões, rótulos de campos e títulos.

Conforme o comportamento de algorítimos de ranqueamento apontado por Page et al. 1999 que trabalham analisando conteúdo exposto em uma página *web* para definir a posição que será listada no motor de busca e visto que este é um fator fundamental segundo Setiawan et al. 2020 para se ganhar relevância na divulgação ou competitividade, foi necessária a evolução do projeto para uma nova versão que melhorasse este aspecto.

Outro problema presente na versão piloto da aplicação CMS estática era uma deficiência da usabilidade por parte do usuário final. Antes do desenvolvimento dessa primeira versão, para gerenciar o conteúdo da página *web* da conferência, o usuário precisava alterar o arquivo estático JSON manualmente. Em uma pequena evolução, foi desenvolvido um painel dashboard, uma *interface* gráfica para a manipulação dos dados deste arquivo JSON estático. Para conseguir executar essa operação, era necessário o *download* dessa *interface* e uma configuração de ferramentas para que a ferramenta funcionasse localmente na máquina do usuário.

A dificuldade em manipular manualmente um arquivo JSON ou a instalação e configuração das ferramentas necessárias para a execução da *interface* gráfica de manipulação deste arquivo resulta na exclusão do uso da ferramenta por parte do conjunto de usuários que possuem média ou baixa familiaridade com tecnologia, limitando assim o alcance do uso da ferramenta reduzindo a atuação do objetivo para qual o projeto foi desenvolvido: facilitar para entidades acadêmicas a divulgação de eventos científicos.

4.2 Metodologia de Desenvolvimento da Aplicação

Objetivando a qualidade do produto, redução de falhas e documentação do projeto, foi adotado uma metodologia de desenvolvimento de *software*, o Kanban. As histórias e tarefas de história foram divididas e planejadas com base nos requisitos existentes para o projeto.

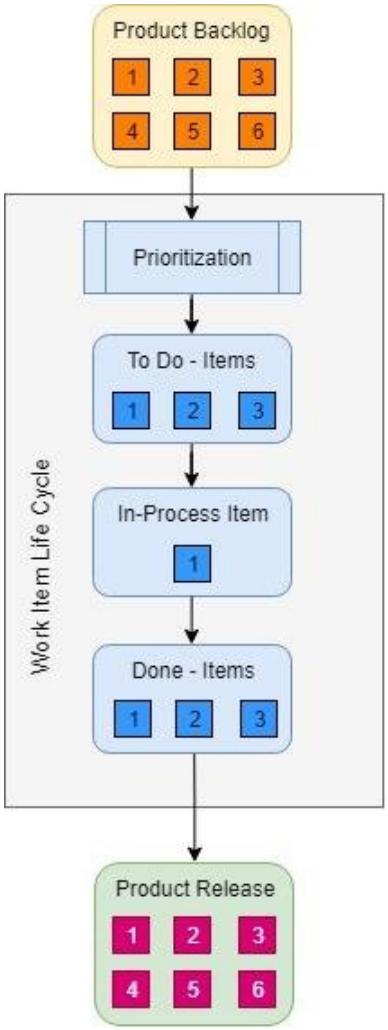
4.2.1 Kanban

O Kanban é uma metodologia de gestão de fluxo de projeto de desenvolvimento de produtos. Originalmente, se amparava em quadros com informação disposta visualmente, mas com a evolução dos processos de desenvolvimento Figura 4.3 (Bhavsar, Shah e Gopalan 2020), diversas plataformas virtuais implementaram e automatizaram em determinadas tarefas referentes a essa gestão.

O gerenciamento do fluxo do processo de desenvolvimento da aplicação foi executado de forma totalmente virtual em um quadro criado no projeto do *GitHub* (https://github.com/LucasFonsecadosSantos/Open-Stacom/projects/1). A plataforma cliente GitHub possui mecanismos de automatização do Kanban por meio do controle de *Issues* do projeto, capaz de associar *commits* e *pull-requests* a cada história ou tarefa referenciada nos *cards*.

A plataforma utilizada dispõe de forma pré-configurada determinados padrões de quadro para o controle do fluxo, todos eles no formato *To do*, *In progress*

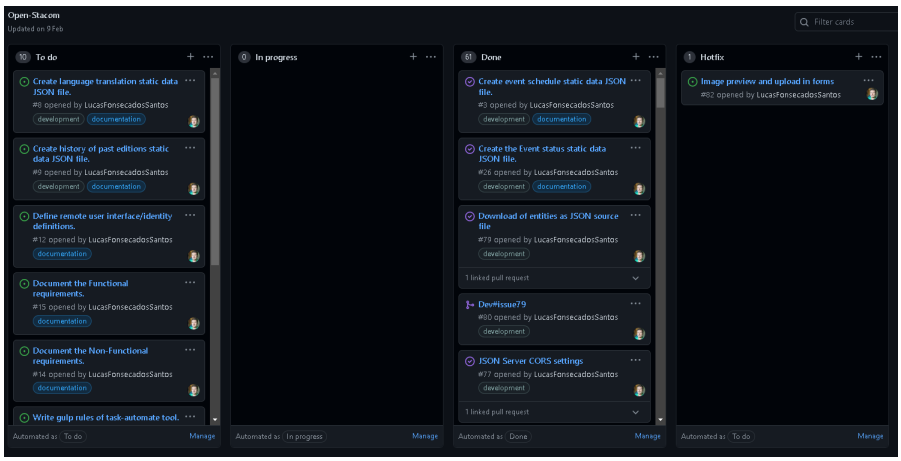
Figura 4.3 – Gestão de fluxo do Kanban.



Fonte: (Bhavsar, Shah e Gopalan 2020)

e *Done*, variando na automatização da atualização do estado dos *cards* a partir de *issues*, *pull-requests* ou de forma manual. O usuário da plataforma também possui liberdade para adicionar, remover ou editar qualquer uma das colunas descritas. Neste projeto, foi adicionado uma coluna de *Hotfix* para comportar histórias e tarefas testadas, mas que apresentaram alguma categoria de erro com necessidade de correção, conforme ilustrado na captura de tela abaixo Figura 4.4:

Figura 4.4 – Kanban do projeto.



Fonte: Autor

O quadro do Kanban foi programado para ser atualizado conforme o estado das *issues* registradas no projeto são modificados. Assim sendo, todas as histórias e tarefas estão documentadas no projeto por via das *issues*.

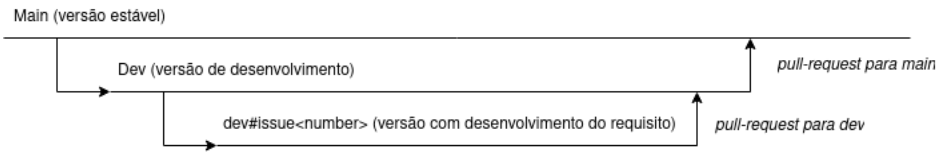
4.2.2 Controle de Versão

O controle de versão executado no projeto utilizou a ferramenta livre e *open-source* GIT com *client* GitHub da Microsoft. O fluxo de desenvolvimento foi padronizado com base em diferentes *branches* para controlar os estados do projeto. No fluxo, é considerado uma *branch main* com a versão estável da aplicação. No próximo nível, uma *branch* nomeada como dev, contém a versão em fase desenvolvimento da aplicação. Para além dessas, cada história deve possuir uma *branch* própria, com padrão de nomenclatura dev#issue<numero_issue>. As modificações construídas nas *branches* de história devem ser *mergeadas*, após devidamente testadas e validadas funcionalmente, com a *branch main*.

O padrão determina que para correção de *bugs* em histórias, deverá ser criada uma *branch* com nomenclatura hotfix#issue<numero_issue>.

Segundo metodologia definida, todo *pull-request* deve ser devidamente documentado com a *issue* associada, ou seja, deverá estar associado a uma história finalizada. Apesar de o projeto conter somente 1 (um) desenvolvedor, o processo de *merge* entre diferentes *branches* deve ser concluído somente após uma etapa revisão de código utilizada para validar requisitos funcionais e não funcionais referentes a história em questão bem como definições técnicas de implementação objetivando a redução de *bugs* na versão presente na *branch* dev e consequentemente em uma futura versão estável do produto (presente na *barnch main*) Figura: 4.5.

Figura 4.5 – Esquema de branches utilizado.



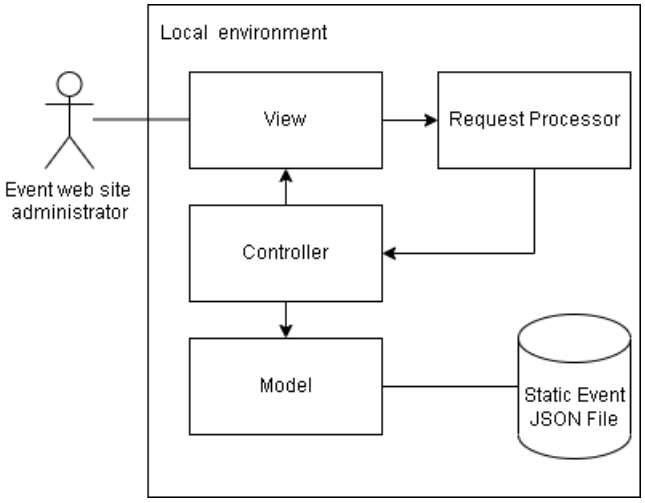
Fonte: Autor

4.3 Arquitetura

Foram implementados diferentes modelos arquiteturais na aplicação. A versão piloto do projeto, possuía somente uma camada *frontend* executada localmente na máquina cliente. O projeto consistia em uma *interface* cujo *download* era feito pelo usuário. Essa *interface* era utilizada para administrar o conteúdo referente ao evento de conferência acadêmico armazenado em um arquivo estático de formato JSON.

Essa *interface* local foi construída em arquitetura MVC (Model-View-Controller) onde todas as operações de CRUD (*Create, Read, Update e Delete*) e regras de negócio estavam contidas neste modelo arquitetural conforme consta na Figura 4.6. A implementação nessa versão não estava construída sobre nenhuma categoria de *Framework*, utilizado linguagem Javascript pura.

Figura 4.6 – Esquema arquitetural da *interface* de administração de conteúdo Open-Stacom na versão 1.0.



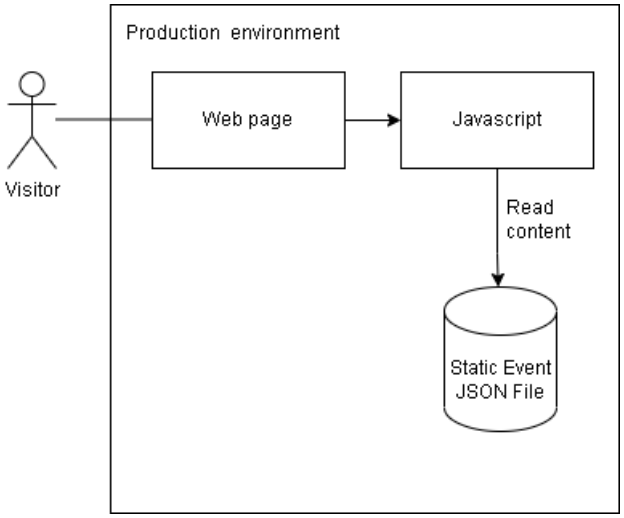
Fonte: Autor

Na primeira versão, não havia uma arquitetura cliente servidor implementada que permitisse alguma categoria de operação sobre o conteúdo já publicado, então, para versão 1.0, era necessário o *deploy* do arquivo estático JSON com todo o conteúdo referente ao evento em ambiente de produção para atualização dos conteúdos. A página *web* publicada possuía funções simples que executavam requisições assíncronas Ajax para o carregamento da informação e apresentação na *interface* do *website* conforme consta na Figura: 4.8.

4.4 Ambiente

A aplicação Open-Stacom desde o projeto-piloto, em sua primeira versão, foi projetada para ser simples e não necessitar de uma infraestrutura complexa ou custosa (na perspectiva computacional e financeira), tanto a aplicação quanto sua saída resultante do processamento, a página *web* estática para divulgação do evento. Essa característica se deve ao contexto de seu nascimento, discutido no capítulo de introdução deste trabalho.

Figura 4.7 – Carregamento de conteúdo estático na página *web* Open-Stacom na versão 1.0.

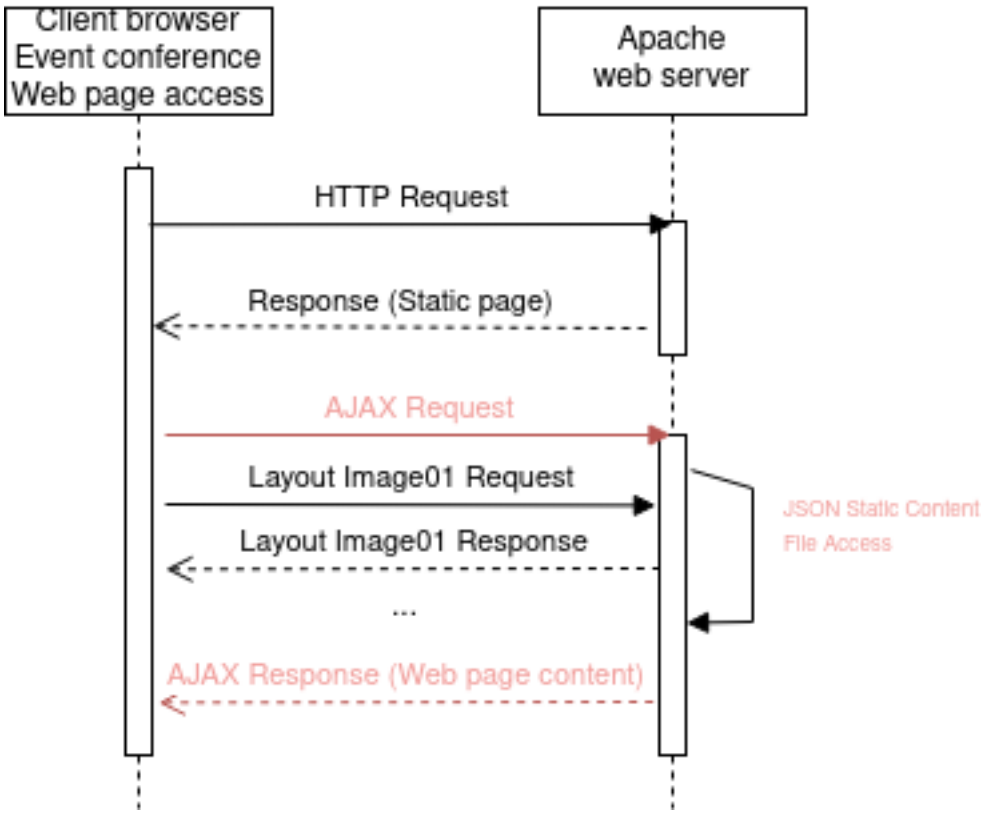


Fonte: Autor

A evolução do projeto, entretanto, trouxe um aumento da necessidade de infraestrutura para comportar a aplicação Open-Stacom como um serviço web, mas a página *web* gerada como saída do processamento da aplicação, não necessitou de qualquer ajuste nestes requisitos para seu pleno funcionamento. Para execução da aplicação Open-Stacom como um serviço web, é necessário a disponibilização de um ambiente com serviços NodeJS, Angular e Java Spring;

Já o desenvolvimento do projeto ocorreu em ambiente com sistema operacional GNU/Linux Arch-Linux, utilizando a IDE (*Integrated development environment* ou Ambiente de desenvolvimento integrado) Intellij IDEA *Community Edition*, *Visual Studio Code* com *Browser* Mozilla Firefox, um serviço JsonServer NodeJS, Angular e Apache Tomcat. Também foi utilizado para análise de requisições a ferramenta Postman.

Figura 4.8 – Diagrama de sequência do carregamento de conteúdo estático na página web Open-Stacom na versão 1.0.



Fonte: Autor

4.5 Linguagem e *Frameworks*

Devido ao levantamento dos novos requisitos não funcionais como melhoria das condições de SEO/SERP na página *web* gerada de apresentação da conferência gerada pela aplicação Open-Stacom, foi necessário a adoção de alguns *frameworks* de desenvolvimento para se facilitar o processo de evolução do *software*.

4.5.1 Angular

A camada *frontend* do projeto basicamente conta com duas partes separadas: a *interface* de administração do conteúdo do evento, o *dashboard* e a página

web produto gerada pelo processamento da aplicação. A tecnologia da página *web* gerada não é relevante para a discussão, visto que seu desenvolvimento é baseado em *templates*, podendo inclusive ser desenvolvida por terceiros. Devido a este aspecto, a tecnologia utilizada para a construção dos *templates* pode ser diversas como Angular, React, dentre outras.

Já o *framework* adotado para o *dashboard* de administração do conteúdo é o Angular em sua versão 12.1.2 com serviço NodeJS em sua versão 17 e gerenciador de pacotes NPM verão 8.5.4, conforme Figura: 4.9

4.5.2 Java Spring

Na camada *backend* da aplicação que na primeira versão do *software* não existia, foi utilizado o framework Java Spring Boot com bibliotecas específicas para injeção do conteúdo da página *web* a ser gerada cuja discussão ocorrerá no próximo capítulo deste trabalho. Foi utilizado versão 2.6.1 no Java Spring Boot e versão 11 do Java.

Figura 4.9 – Versão do framework Angular utilizado na camada de *frontend* do dashboard e seus pacotes.

Fonte: Autor

5 PROPOSTA

Constatou-se durante o uso da primeira versão do *software* Open-Stacom a necessidade de uma série de mudanças, modificações essas que tornaram necessária uma refatoração arquitetural impactante em todo o projeto.

5.1 Novos requisitos

Uma das necessidades levantadas refere-se a uma melhoria de SEO/SERP. Conforme abordado no capítulo 2 deste trabalho, alguns dos algorítimos dos buscadores responsáveis pelo ranqueamento das páginas na lista de resultados da pesquisa de um determinado usuário efetua uma análise de conteúdo presente na página relacionando-o com as palavras chaves pelo usuário informado Page et al. 1999.

A primeira implementação do projeto produzia como resultado de seu processamento, uma página *web* que carregava o conteúdo a ser exibido de forma assíncrona através de uma requisição Ajax conforme apresentado no capítulo anterior. Isso destaca o primeiro problema: toda a estrutura da página é renderizada pelo seu buscador sem que o conteúdo ainda esteja carregado por completo, isso significa, que a página antes de ser carregada, possui somente sua estrutura de *layout* declarada em HTML sem qualquer categoria de conteúdo, impossibilitando a análise de conteúdos por parte dos robôs e algorítimos responsáveis pelo ranqueamento segundo este quesito.

A vantagem desta categoria de implementação é o ganho de desempenho, no aspecto de carregamento da página, dado que o usuário, durante seu acesso, não necessariamente precisa aguardar todo o conteúdo ser carregado para poder navegar pela *interface* gráfica renderizada, entretanto isso resulta em uma perda da otimização para mecanismos de busca, fazendo com que a página perca muitas posições na listagem do buscador.

Essa perda de posições é extremante relevante segundo aponta Zhu e Wu 2011, conforme apresentado no capítulo 2 deste trabalho resulta na queda de acessos e interesse, consequentemente no fracasso da divulgação do evento.

Para contornar este problema, era necessário que a página *web* gerada pelo sistema tivesse todo seu conteúdo injetado no código HTML de forma estática, em um processamento executado durante sua geração. Para tal, foi necessário a adoção de uma nova camada ao projeto, a camada *backend*.

Também constituía-se como novo requisito, a disponibilização da aplicação como uma plataforma *online* onde o usuário acessa o site do *software* e inicia o fluxo de operação. Outrora, na primeira versão do Open-Stacom, era necessário que o usuário efetuasse um *download* de toda a ferramenta para que a execução fosse realizada em ambiente local. O objetivo deste requisito era facilitar o uso por parte do usuário, que bastaria acessar o site da ferramenta para configurar seu evento e gerar a página web desejada.

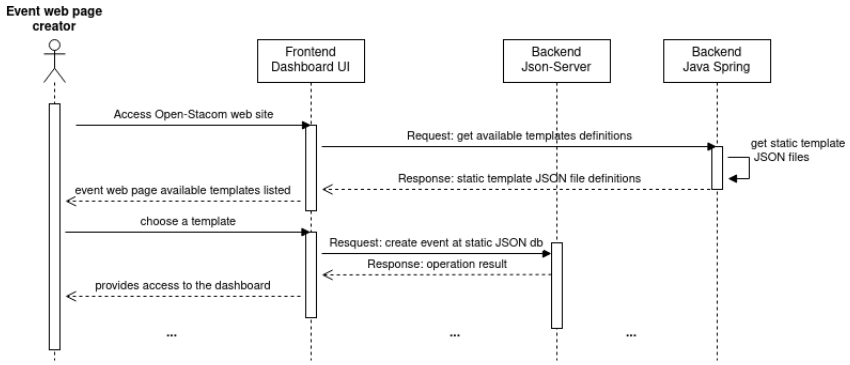
5.2 Fluxo de Operação

Objetivando adequação ao novo requisito para disponibilizar o *software* em um serviço *online*, foi implementada uma nova arquitetura de serviços que impactou diretamente no fluxo de operação dos usuários segundo ilustra a Figura 5.1 com a nova abordagem cliente-servidor da aplicação.

Basicamente, um usuário que objetiva gerar uma página para divulgação de seu evento, deve acessar o site da plataforma, selecionar um *template* disponível e através do *dashboard*, adicionar, atualizar ou remover qualquer informação que deseja publicar em sua página como cronograma de atividades, lista de comitês, lista de patrocinadores, dentre outas informações.

Após preencher os dados, o usuário poderá acionar a geração da página. O sistema então valida todas as informações inseridas perante as definições estabelecidas pelo *template* escolhido e injeta essas informações na página HTML,

Figura 5.1 – Diagrama de sequência para operação de acesso e criação de um novo evento.



Fonte: Autor

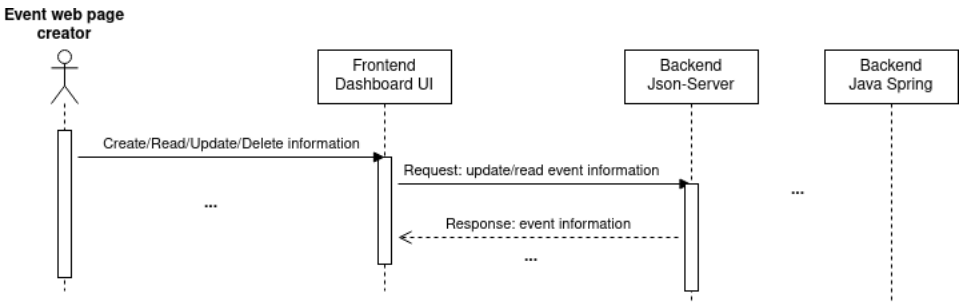
retornando ao usuário uma página estática pronta para ser hospedada em qualquer servidor *web*.

Foi implementada uma arquitetura multicamadas simples com 3 (três) serviços paralelos. O primeiro serviço, uma aplicação Angular (*frontend*), responsável por fornecer uma *interface* gráfica de visualização e atualização das informações do evento. Também é responsável pela validação dos dados em um primeiro nível.

Já o segundo serviço, implementado em NodeJS, é uma aplicação Json-Server que pretende somente manter o estado do evento a ser configurado pelo usuário. A aplicação acessa e atualiza informações do evento em questão armazenadas em um arquivo de formato JSON. Frequentemente, durante a operação de administração do conteúdo do evento, a aplicação Angular está comunicando com este serviço, sempre para ler e atualizar os dados do evento em questão, conforme demonstra a Figura 5.2.

O terceiro serviço mencionado é a camada *backend* implementada sob *framework* Java Spring. Essa camada nasceu para assumir a responsabilidade do processo de geração de código no projeto. Sua função é efetuar uma segunda análise de validação sobre os dados fornecidos via requisição da camada de *frontend*

Figura 5.2 – Diagrama de sequência para operação de acesso e atualização de um evento.



Fonte: Autor

administrados pelo usuário e a injeção/processamento destes dados nos arquivos em formato HTML pertencentes ao *template* pré definido.

Após a conclusão das inserções, atualizações e remoções de dados, o usuário pode acionar a geração da página *web*. Agora, a aplicação Angular envia uma requisição a camada *backend* Java Spring. Em ordem, a controladora responsável nessa camada aciona as rotinas de validação da informação que chega e em seguida, o serviço de geração de código.

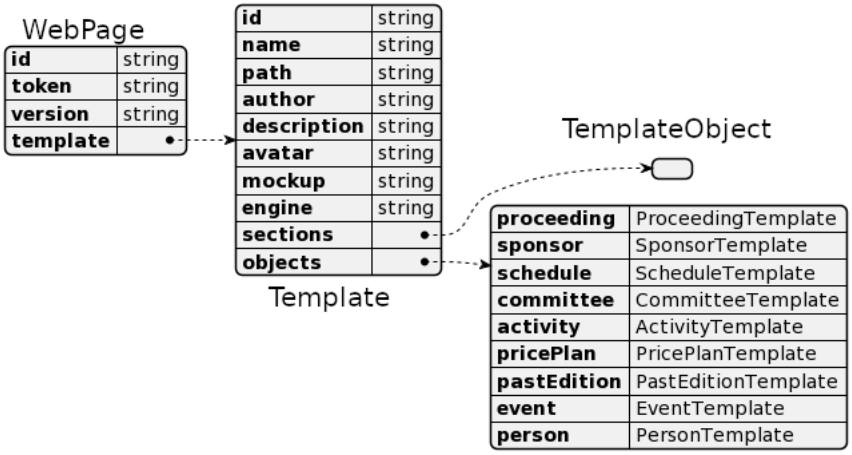
Este serviço de geração de código implementado varia conforme o motor de renderização definido no *template* escolhido. Foram adicionadas ao projeto as bibliotecas Thymeleaf, Freemarker, Groovy, Jade and Java Server Pages, possibilitando que o criador do *template*, optasse por diferentes *guidelines* no momento de construir seu código, visto que cada motor de *template*, possui sua própria sintaxe junto ao código HTML.

5.3 Modelagem de dados

Conforme explanado nas seções anteriores, todos os dados referentes ao evento e entidades relacionadas são armazenadas de forma estática em um arquivo de formato JSON durante a execução da operação do usuário na aplicação OpenStacom até o momento de geração da página *web*. Sendo assim, o conteúdo da

página *web* do evento gerenciado pela aplicação é definido conforme o diagrama abaixo na Figura: 5.3:

Figura 5.3 – Diagrama de entidades que representam o conteúdo de uma página web gerada pelo software Open-Stacom.



Fonte: Autor

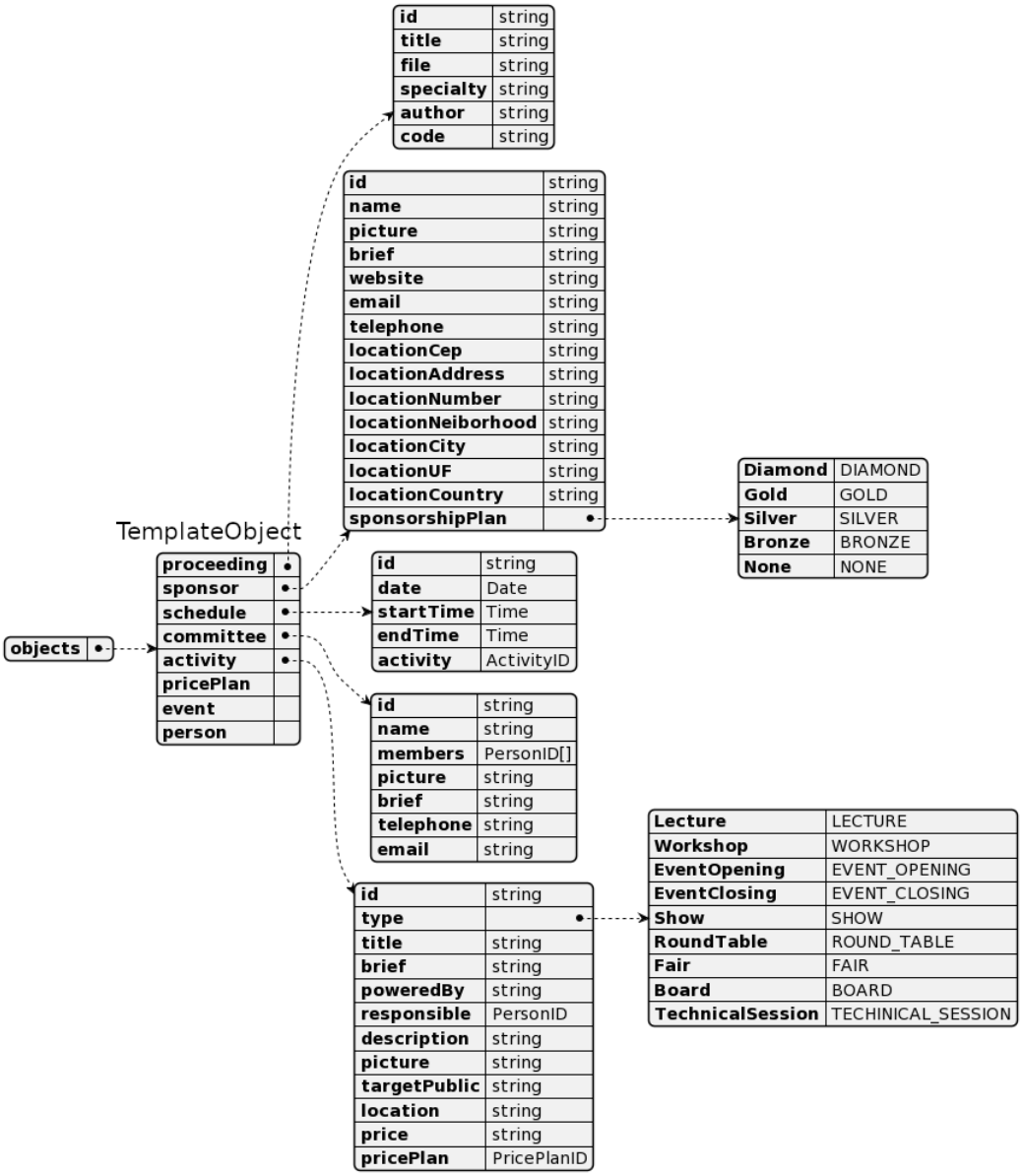
O objeto *template* possui alguns atributos especiais que o define sua informação quanto ao seu desenvolvimento. Atributos como *"author"* e *"engine"* descrevem informações que permitem o rastreio sobre seu desenvolvimento, visto que poderá ser fruto da contribuição de terceiros via repositório disponibilizado no cliente Github.

O atributo *"engine"* no objeto *Template* informa qual será o motor de *template* (*template engine*) que será utilizado para efetuar o processamento da página *web* de divulgação do evento, abordagem está, que será discutida na próxima seção 5.4.

O objeto *Template* também contem um atributo *"objects"* com diversas outras entidades, conforme abaixo ilustrado no diagrama da Figura 5.4 e na Figura 5.5. Cada entidade documentada também possui definições estruturais do *template*, abstraído por um objeto *TemplateField* que define um tamanho máximo para a sequência de caracteres do conteúdo, um tamanho mínimo e sua obrigatori-

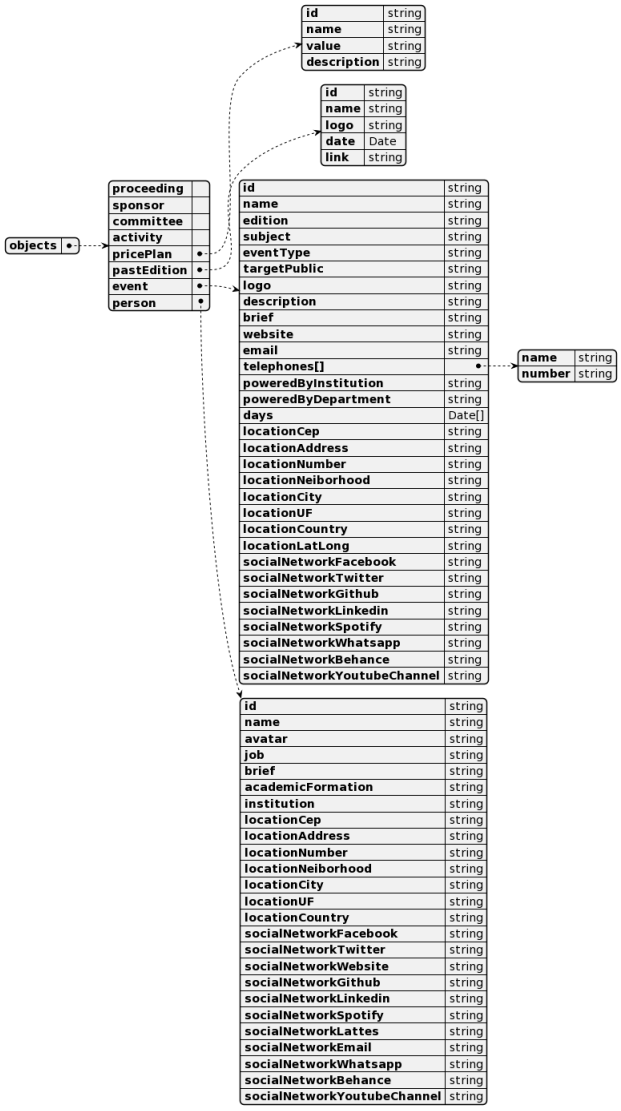
edade, para que o conteúdo inserido pelo usuário se apresente de forma adequada na página *web template*. Para fins de mapeamento documental, abaixo está ilustrado somente a modelagem de dados referente ao conteúdo e não a definição do *template*.

Figura 5.4 – Diagrama de entidades parte de um *Template*.



Fonte: Autor

Figura 5.5 – Diagrama de entidades parte de um *Template*.



Fonte: Autor

5.4 Arquitetura

Para a implementação dos requisitos levantados, foi determinada a construção de uma API na camada *back-end* responsável pela geração da página *web*

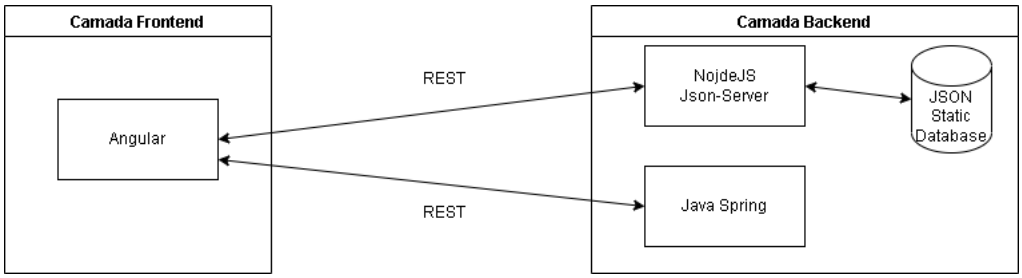
com os serviços necessários para a injeção e validação de todo o conteúdo da página do evento.

Além disso, a API *backend*, desenvolvida utilizando o *framework* Java Spring, também gerencia os *templates* disponíveis para o usuário.

No projeto, consta uma segunda API utilizando tecnologia NodeJS com a biblioteca Json-Server que tem como responsabilidade principal, armazenar o estado do evento enquanto o usuário prossegue em seu fluxo de operação no *dashboard*, inserindo, editando, buscando ou removendo informações de conteúdo. Este estado da entidade evento é armazenado em um arquivo temporário em formato JSON consumido por essa API em operações de CRUD pela camada *frontend* (aplicação angular do *dashboard*).

Basicamente, disponível ao usuário como um serviço web, a arquitetura do sistema, em uma perspectiva macro, é constituída dos componentes ilustrados na Figura 5.6.

Figura 5.6 – Diagrama geral da implementação do *software* Open-Stacom.



Fonte: Autor.

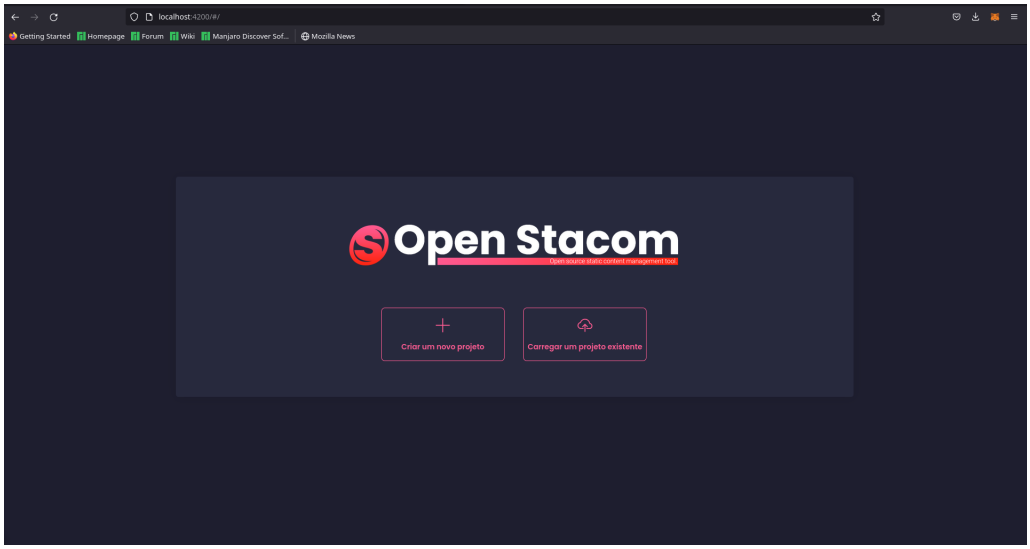
5.4.1 Arquitetura da Camada Frontend

Na camada *frontend* da aplicação, a implementação conta com o *framework* Angular. Por padrão, o Angular implementa o modelo arquitetural MVC (*Model-View-Controller*) composto de uma estrutura de componentes e serviços. Para cada entidade presente no sistema, 4 (quatro) serviços são implementados, cada um para uma determinada operação de CRUD.

Estes serviços de CRUD na camada *frontend* também operam para manter a consistência dos dados referente aos relacionamentos existentes entre as entidades. Cada entidade possui um identificador numérico único (ID) representado seu registro. Como os dados são persistidos em arquivos de formato JSON sem o controle de nenhum sistema de banco de dados, então, foi necessária a implementação de dos relacionamentos entre as entidades. As regras de negócio responsáveis por tais definições, que em sua natureza, traduzem operações *join* estão contidas em serviços de CRUD construídas em *services* do Angular.

A aplicação *frontend* disponibiliza ao usuário a *interface* gráfica de seleção de *template* ilustrada na Figura 5.7, *interface* de criação de um evento demonstrada na Figura 5.8 e do *dashboard* conforme apresenta a Figura 5.9 do Open-Stacom que fornece uma visualização completa dos dados bem como fornece mecanismos de entrada e atualização destes dados referentes ao evento.

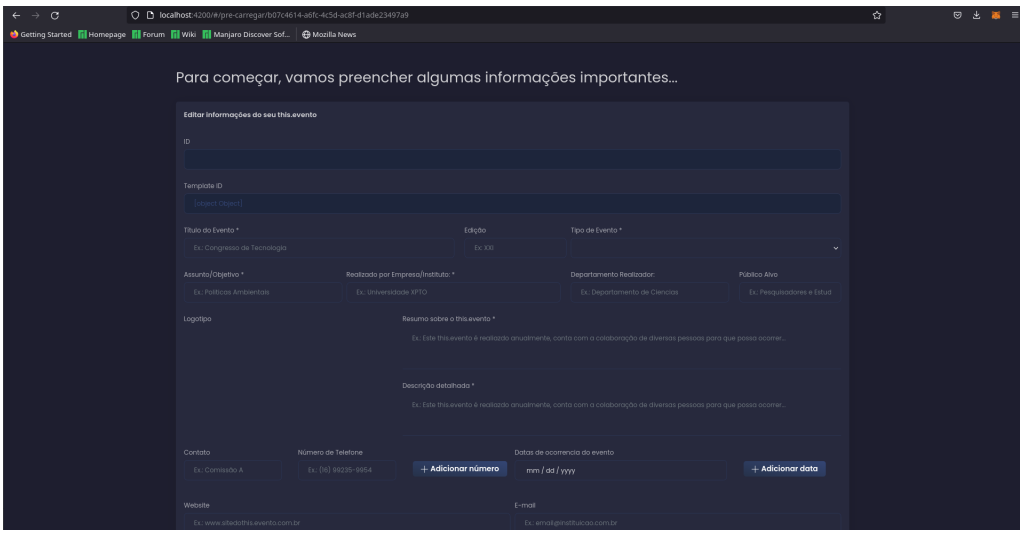
Figura 5.7 – *Interface* de seleção de *template*.



Fonte: Autor.

Na *interface* de seleção de *template*, o usuário visualiza uma listagem de opções disponíveis pela aplicação para a página do evento e pode decidir qual será utilizada.

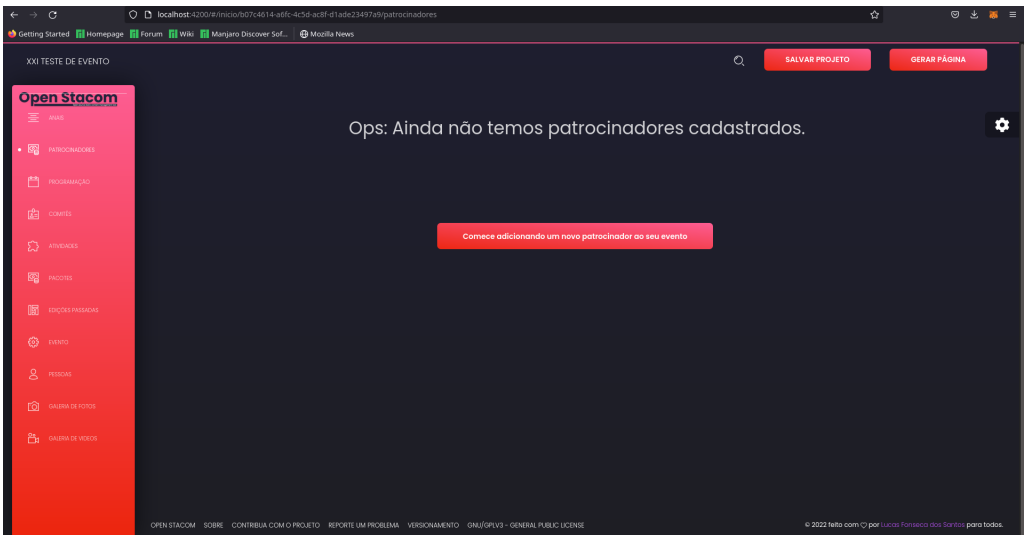
Figura 5.8 – *Interface* de criação de um evento.



Fonte: Autor.

Após a seleção do *template* que será utilizado, o usuário deverá preencher informações básicas do evento. Essas informações são pré-definidas pela aplicação Open-Stacom. A quantidade de caracteres e a obrigatoriedade de um determinado dado é definida pelo *template* escolhido.

Figura 5.9 – *Interface* do dashboard.



Fonte: Autor.

Após o preenchimento dos dados básicos, o usuário ganha acesso a um *dashboard* onde poderá administrar todo o conteúdo da página que apresentará o evento.

Conforme descrito na seção anterior 5.2, essa camada se comunica com um serviço NodeJS e outro Java Spring, ambos pertencentes a camada de *backend* do sistema. A comunicação entre essas três camadas é baseada em uma arquitetura REST, com tráfego dos dados totalmente baseados em requisições com *data* em formato JSON.

5.4.2 Arquitetura da Camada *Back-end*

Open-Stacom conta com duas diferentes aplicações em camada *back-end*. A primeira aplicação, desenvolvida em NodeJS simplesmente abriga a implementação e a disponibilização de uma biblioteca Node chamada Json-Server, não apresentando nenhuma categoria de organização arquitetural. Apresenta-se somente em um conjunto de *endpoints* definidos pela própria biblioteca Json-Server com base na estrutura do arquivo estático JSON utilizado como base de dados.

A segunda aplicação pertencente a camada *back-end* é desenvolvida utilizando *framework* Java Spring com módulo Web, disponibilizando uma arquitetura de implementação MVC com controladoras REST. Entretanto, não há neste conjunto a implementação da camada de visualização (*View*), visto que os dados para visualização são retornados para a aplicação a camada *frontend* em formato *data* JSON como dados brutos, sendo a responsabilidade de apresentação destes, delegada a aplicação Angular do *frontend*.

Sob responsabilidade da aplicação Java Spring, está um conjunto de validações sobre os dados perante as definições de *template* pré estabelecidas como, por exemplo, tamanho de campos e obrigatoriedade. Além disso, o processamento dos arquivos de *template* das páginas com a injeção dos dados sobre o evento fornecidos pelo usuário estão implementados nessa aplicação.

Na atual versão desenvolvida, a camada do Java Spring recebe somente duas requisições do usuário por via da camada *front-end*: A seleção/escolha de um *template* que será utilizado na página para apresentação do evento e a requisição para se processar o *template* com base no conjunto de dados administrado pelo usuário, dados estes recebidos através de um padrão REST. Como response, essa camada retorna ao usuário o objeto JSON que contém a definição do *template* escolhido, no caso do primeiro fluxo operacional citado e uma URL para *download* de um arquivo compactado em formato ".zip"contendo a página em formato HTML com todos os dados do evento injetados em sua estrutura no momento do processamento.

A adoção de padrões estruturais de projeto nessa camada permitiram a implementação de mais de um motor de *template* para o processamento da página, aumentando a quantidade de *guidelines* que desenvolvedores terceiros que estejam interessados em contribuir com o projeto Open-Stacom possam seguir no momento de desenvolver novos *templates* para o *software*. Para os desenvolvedores, as principais diferenças entre as *guidelines* estão na sintaxe e em mecanismos de controle de fluxo.

5.5 Resultado do processamento

Para o usuário final que administra o conteúdo de um evento através da plataforma Open-Stacom, é retornado um arquivo compactado ".zip"ao término de sua operação com os arquivos em formato HTML e imagens. Este arquivo HTML contem todo o conteúdo referente ao evento já injetado em sua estrutura que será renderizado em um aplicativo *browser* de forma estática.

Um arquivo HTML com conteúdo apresentado de forma estática possui um ganho na velocidade de processamento por parte da execução no *Browser* bem como reduz, em grande escala, os requisitos necessários para seu funcionamento como ambiente. Para a hospedagem dessa página estática, não é necessário um

ambiente com base de dados ou grandes configurações de infraestrutura como memória e processamento, reduzindo assim, custos computacionais e consequentemente, financeiros para sua hospedagem e disponibilização na *internet*.

6 CONCLUSÃO

O objetivo deste trabalho buscou sanar os problemas sistêmicos referentes ao SEO e usabilidade apresentados apresentando processo de implementação destes novos requisitos em uma nova versão da aplicação Open-Stacom para geração de páginas *web* estáticas para a divulgação de evento acadêmico científicos.

Com a implementação do modelo arquitetural discutido e a modelagem dos dados que representam o evento, foi possível a construção de um mecanismo de injeção de conteúdo na *interface* gráfica da página *web* de forma estática. Desta forma, o conteúdo não faz parte mais de um carregamento dinâmico na página após a renderização dos elementos estruturais, mas sim, faz parte dos próprios elementos estruturais, eliminando o problema da ausência de insumos (conteúdo) necessários para avaliação por meio dos algorítimos de ranqueamento de páginas dos motores de busca.

Com a adoção de uma camada *back-end* para o processamento de injeção de dados na página *web* gerada e o modelo arquitetural REST, a aplicação se tornou um serviço *online* disponível ao usuário, reduzindo assim dificuldades quanto a instalação e execução local. Esse aumento da facilidade do uso (melhoria da usabilidade), acarreta um maior alcance de público, indivíduos com baixa familiaridade com tecnologia, não encontrará maiores dificuldades para atingir seu objetivo na operação.

6.1 Trabalhos Futuros

Entende-se como possibilidade para trabalhos futuros possíveis evoluções arquiteturais e funcionais para a aplicação. Migração para padrões de maior escalabilidade como a implementação de um modelo arquitetural de microsserviços, funcionalidades de *interface* para melhorar a experiência do usuário no uso do sistema bem como a internacionalização do idioma empregado no sistema.

A adição de novos *templates* de páginas *web* para eventos também pode ser abstraída como possíveis trabalhos futuros na evolução do projeto.

REFERÊNCIAS

Angeli et al. 2003 ANGELI, A. D. et al. On the advantages of a systematic inspection for evaluating hypermedia usability. *International Journal of Human-Computer Interaction*, Taylor & Francis, v. 15, n. 3, p. 315–335, 2003.

Bevan 1998 BEVAN, N. *European Usability Support Centre*. 1998.

Bevan, Claridge e Petrie 2005 BEVAN, N.; CLARIDGE, N.; PETRIE, H. Tenuta: simplified guidance for usability and accessibility. In: CITESEER. *Proceedings of HCI International*. [S.l.], 2005.

Bhavsar, Shah e Gopalan 2020 BHAVSAR, K.; SHAH, V.; GOPALAN, S. Scrumbanfall: an agile integration of scrum and kanban with waterfall in software engineering. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, v. 9, n. 4, p. 2075–2084, 2020.

Dorosh e Kuchmij 2009 DOROSH, O.; KUCHMIJ, N. Designing of e-commerce system by cms joomla software. In: *2009 10th International Conference - The Experience of Designing and Application of CAD Systems in Microelectronics*. [S.l.: s.n.], 2009. p. 400–400.

EasyChair Conference System EASYCHAIR Conference System. Disponível em: <http://www.easychair.org/>.

Garlan 2014 GARLAN, D. Software architecture: a travelogue. In: *Future of Software Engineering Proceedings*. [S.l.: s.n.], 2014. p. 29–39.

Insights 2013 INSIGHTS, G. T. *How to Make Your Website Work Across Multiple Devices*. 2013.

Iso 1998 ISO, W. 9241-11. ergonomic requirements for office work with visual display terminals (vdts). *The international organization for standardization*, v. 45, n. 9, 1998.

Krug 2001 KRUG, S. *Nao me faça pensar. Traduçao de Roger Maioli dos Santos*. [S.l.]: Sao Paulo: Market Books, 2001.

Lo, Phan e Goi 2007 LO, S.-W.; PHAN, R. C.-W.; GOI, B.-M. On the security of a popular web submission and review software (wsar) for cryptology conferences. In: SPRINGER. *International Workshop on Information Security Applications*. [S.l.], 2007. p. 245–265.

Martin 2019 MARTIN, R. C. *Arquitetura Limpa: O guia do artesão para estrutura e design de software*. [S.l.]: Alta Books Editora, 2019.

Martins et al. 2012 MARTINS, A. I. et al. The international classification of functioning, disability and health as a conceptual model for the evaluation of environmental factors. *Procedia Computer Science*, Elsevier, v. 14, p. 293–300, 2012.

Martins et al. 2013 MARTINS, A. I. et al. Avaliação de usabilidade: uma revisão sistemática da literatura. *Revista Ibérica de Sistemas e Tecnologias de Informação*, Associação Ibérica de Sistemas e Tecnologias de Informacao, n. 11, p. 31, 2013.

Nielsen 1994 NIELSEN, J. *Usability engineering*. [S.l.]: Morgan Kaufmann, 1994.

Page et al. 1999 PAGE, L. et al. *The PageRank Citation Ranking: Bringing Order to the Web.* [S.l.], 1999. Previous number = SIDL-WP-1999-0120. Disponível em: <http://ilpubs.stanford.edu:8090/422/>.

Patel, Rathod e Parikh 2011 PATEL, S. K.; RATHOD, V.; PARIKH, S. Joomla, drupal and wordpress - a statistical comparison of open source cms. In: *3rd International Conference on Trendz in Information Sciences Computing (TISC2011)*. [S.l.: s.n.], 2011. p. 182–187.

Setiawan et al. 2020 SETIAWAN, A. et al. The optimization of website visibility and traffic by implementing search engine optimization (seo) in palembang polytechnic of tourism. *CommIT (Communication and Information Technology) Journal*, v. 14, n. 1, p. 31–44, 2020.

Soegaard e Dam 2012 SOEGAARD, M.; DAM, R. F. The encyclopedia of human-computer interaction. *The encyclopedia of human-computer interaction*, The Interaction Design Foundation, 2012.

Valente 2020 VALENTE, M. T. Engenharia de software moderna. *Princípios e Práticas para Desenvolvimento de Software com Produtividade*, v. 1, 2020.

Wiener, Ekholm e Haller 2015 WIENER, L.; EKHOLM, T.; HALLER, P. Modular responsive web design using element queries. *arXiv preprint arXiv:1511.01223*, 2015.

Wongsalam e Senivongse 2019 WONGSALAM, S.; SENIVONGSE, T. Visual design and code generation of user interface based on responsive web design approach. In: *Proceedings of the 2019 3rd International Conference on Software and e-Business*. [S.l.: s.n.], 2019. p. 51–59.

Zhu e Wu 2011 ZHU, C.; WU, G. Research and analysis of search engine optimization factors based on reverse engineeing. p. 225–228, 2011.

APÊNDICE A – Como contribuir com o projeto Open-Stacom?

Open-Stacom é um *software* livre e *open-source* e contribuições constituem uma parte extremamente importante no ciclo de evolução do projeto. Neste aspecto, conhecer as *guidelines* é necessário para o desenvolvedor que deseja contribuir com a construção da aplicação.

.1 Procedimentos de contribuição da comunidade

Para contribuir com o projeto, todas as modificações devem ser submetidas via *pull-request* para a "contrib". Nessa *branch*, as modificações são validadas pelos desenvolvedores oficiais associados ao projeto para que posteriormente possam ser integradas ao projeto.

Modificações são validadas com base em definições técnicas e funcionais, ou seja, as modificações devem estar enquadradas nos princípios do projeto, licença, objetivos e necessidades levantadas pela comunidade.

Para desenvolver novas funcionalidades ou correções, o repositório oficial deverá ser clonado. Para executar essa operação, o contribuidor deverá utilizar o aplicativo GIT para controle de versionamento. Em ambientes GNU/Linux, o desenvolvedor deverá executar o comando ilustrado no Exemplo 1.

Figura 1 – Comando de clone de repositório na ferramenta GIT

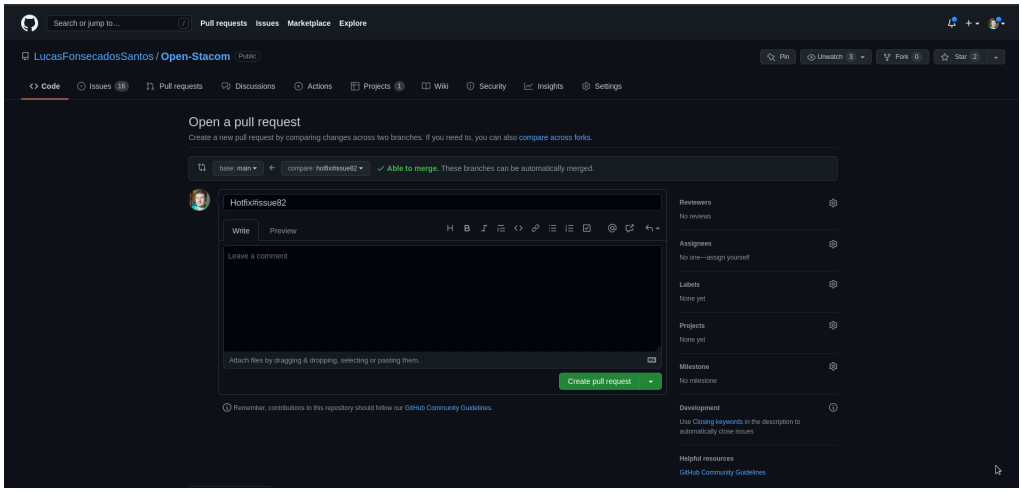
```
1 $ git clone https://github.com/LucasFonsecadosSantos/Open-
    Stacom.git
```

Fonte: Autor

O contribuidor então deverá criar um *pull-request* na *branch* "contrib" preenchendo todas as informações necessárias para identificar as causas e objetivos da modificação proposta, conforme ilustra a Figura 2.

Após aprovadas as modificações por um desenvolvedor oficial do projeto, elas serão incorporadas à versão estável do *software* e o nome do autor das novas implementações deverá ser citado no projeto como contribuidor.

Figura 2 – *Interface* para criação de *pull-request* no cliente Github.



Fonte: Autor.