Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

CPqs Abertas 2.0

provendo a escalabilidade de um sistema Web com White Label via Headless CMS

Fernanda Itoda
Matheus Ribeiro Silva
Lucas de Souza Tenorio
Eike Souza da Silva

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Paulo Meirelles
Cossupervisor: Alfredo Goldman

São Paulo
2023

Lista de abreviaturas

CMS	Sistema de Gerenciamento de Conteúdo (*Content Management System*)
EC2	Computação Elástica na Nuvem (*Elastic Compute Cloud*)
AWS	Serviços da Web da Amazon (*Amazon Web Services*)
RDS	Serviço de Banco de Dados Relacional (*Relational Database Service*)
API	Interface de Programação de Aplicações (*Application Programming Interface*)
IoT	Internet das Coisas (*Internet of Things*)
UI	Interface do Usuário (*User Interface*)
POC	Prova de Conceito (*Proof of Concept*)
VM	Máquina Virtual (*Virtual Machine*)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de figuras

Lista de programas

Sumário

Resumo

Fernanda Itoda
Matheus Ribeiro Silva
Lucas de Souza Tenorio
Eike Souza da Silva. **CPqs Abertas 2.0: *provendo a escalabilidade de um sistema Web com White Label via Headless CMS***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Este trabalho de conclusão de curso visa relatar a evolução e o desenvolvimento do projeto CPqs Abertas ao longo do ano de 2023, apresentando as atividades desenvolvidas, tal como melhorias implementadas, inserção de novas tecnologias e definição de uma nova arquitetura. Na prática, o CPqs Abertas é um portal Web composto por um conjunto de sites das Comissões de Pesquisas (CPqs) com as informações das produções acadêmicas dos institutos da Universidade de São Paulo (USP), obtidas conforme os dados disponíveis por cada docente na Plataforma Lattes do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico). O trabalho teve como objetivo migrar a arquitetura para tornar a inserção de institutos (sites) escalável e prática por meio da estratégia *White Label* via *Headless CMS*, utilizando o framework Strapi. Dessa forma, o maior esforço durante o primeiro ciclo de desenvolvimento foi a adaptação do código legado de obtenção dos dados para funcionar conforme a nova estratégia e tecnologias definidas para esta nova fase do projeto. Com essa nova arquitetura, conseguiu-se unificar os códigos dos sites, eliminando as duplicações existentes, sendo essa uma das melhorias implementadas. Com isso, permitiu-se uma redução de tempo para a inserção de um novo instituto (site), além de evitar erros, devido à replicação do código do site base para criação de um novo instituto na atinga versão do CPQs Abertas . Para validar a estratégia a *White Label* e a nova arquitetura, adicionou-se ao CPqs Abertas um novo instituto, demonstrando que a evolução obtida não só permitirá um trabalho mais ágil para futuros desenvolvedores na inserção de novas unidades, mas também facilita a expansão do projeto como um todo para tentar atender toda a USP.

Palavras-chave: CPqs Abertas. CMS. White Label. Arquitetura de Software.

Abstract

Fernanda Itoda
Matheus Ribeiro Silva
Lucas de Souza Tenorio
Eike Souza da Silva. **CPqs Abertas 2.0:** ***Providing the scalability of a Web system with White Label via Headless CMS***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

This capstone project aims to report the evolution and development of the CPqs Abertas project throughout 2023, presenting the activities carried out, such as implemented improvements, the integration of new technologies, and the definition of a new architecture. CPqs Abertas is a website composed of a collection of sites from Research Committees (CPqs – Comissões de Pesquisas) containing information on academic productions of the institutes of Universidade de São Paulo (USP), obtained from the data available by each professor in the Lattes Platform of the National Council for Scientific and Technological Development (CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico). The work aimed to change the architecture to make the insertion of institutes (sites) scalable and practical through the White Label strategy via Headless CMS, using the Strapi framework. Thus, the significant effort during the first development cycle was adapting the legacy code of data obtainment to fit the new strategy and technologies defined for this new phase of the project. With this new architecture, it was possible reduce time in the context of the insertion of new institutes (sites), besides avoiding mistakes due to code replication of the original site for creating a new institute in the old version of CPqs Abertas. It was added a new institute to the CPqs Abertas to validate the White Label strategy and the new architecture, showing that the obtained evolution will not only enable faster labor for future developers in the insertion of new unities but also make the overall expansion of the project easier, in an attempt to attend USP as a whole.

Keywords: CPqs Abertas. CMS. White Label. Software Architecture.

Capítulo 1

Introdução

Pesquisas e produções acadêmicas são de grande importância para o desenvolvimento da ciência no país. As universidades são protagonistas nesse cenário, pois nelas se encontram o ambiente propício para pesquisa. Nesse contexto, surgiu o projeto CPqs Abertas com o objetivo de servir como vitrine para as produções acadêmicas concretizadas na Universidade de São Paulo (mas podendo ser expandido e adotado por qualquer outra instituições de ensino superior). Além de produzir e desenvolver ciência, é tarefa das universidades brasileiras serem transparentes e externarem o que foi realizado no ambiente científico.

O Projeto CPqs Abertas, inicialmente denominado FAUaberta, teve origem como parte integrante da disciplina "Laboratório de Programação Extrema", ministrada pelo Instituto de Matemática e Estatística (IME) da Universidade de São Paulo (USP) (**tcc2022-cpqsabertas**). Sua concepção emergiu do desejo de ampliar a divulgação da produção intelectual da Faculdade de Arquitetura e Urbanismo (FAU), tornando-a mais acessível e visível para todos na Internet. Após a implementação da página do FAUaberta, surgiu o interesse de outras unidades, sendo elas, o Instituto de Matemática e Estatística (IME) e a Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto (FEARP). Dessa forma, deu-se início ao projeto CPqs Abertas (**tcc2022-cpqsabertas**).

Na fase inicial do projeto, ou seja, ainda como FAUaberta, um componente essencial foi a extração de informações dos currículos dos docentes da FAU via a plataforma Lattes do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico). Isso resultou na criação de um repositório de informações acadêmicas, tornando-se um dos pilares do projeto. Com a entrada do IME e da FEARP, a próxima fase teve como foco a adição de novas unidades. Entretanto, da forma como a arquitetura se encontrava, tais inserções eram trabalhosas e não escalável ao pensar na quantidade de institutos e programa de pós-graduação existentes na USP.

Do ponto de vista de engenharia de software, foi identificado um problema na estratégia adotada para a expansão do projeto, ou seja, na adição de novas unidades, o que na prática significa inserir um novo site no que podemos denominar de plataforma CPqs Abertas. Em resumo, o código-fonte era replicado entre os sites criados para cada instituto. Na prática, copiou-se o código do FAUAberta para conceber os sites do IME e da FEARP no CPQs

Abertas. Em outras palavras, criou-se um novo lote de arquivos com códigos praticamente idênticos, apenas com algumas alterações pontuais no front-end, com dados de cada instituto. Entre as boas práticas de programação, deve-se evitar a duplicação de código, em particular, do ponto de vista de manutenção de software (**martin2008cleancode**).

Nesse cenário, este estudo tem como objetivo aplicar boas práticas de engenharia de software, em especial no contexto das melhores decisões do ponto de vista da arquitetura do projeto para proporcionar a escalabilidade da implantação das páginas das unidades inseridas no CPqs Abertas. Nesse contexto, ao investigar uma solução para o problema de replicação do código base do projeto (FAUAberta), foi identificada a estratégia denominada de "White Label" como uma alternativa para a escalabilidade e manutenção do CPQs Abertas. Do ponto de vista de desenvolvimento de software, em particular de um sistema Web, o White Label é aplicado através de um arcabouço (framework) CMS (*Content Management System* – Sistema de Gerenciamento de Conteúdo).

De maneira geral, um produto "White Label" é fabricado por uma empresa produtora, que vende tal produto para outras; assim, cada empresa coloca a sua marca de forma que o produto aparenta ter sido fabricado pela empresa compradora (**tardi2022**), entre outros motivos, por uma questão de custos e fortalecimento da marca ao ter produtos "próprios". Pensando em um sistema Web, sites de comércio eletrônico são um bom exemplo, pois, uma loja virtual ao ter como base uma plataforma de *E-commerce* já existente, pode manter o seu foco nas vendas, sem grandes preocupações com a manutenção e custo de um sistema exclusivo e próprio.

Originalmente, os arcabouços CMS surgiram como sistemas que auxiliam os usuários a modificar, manter e criar conteúdos de um site, sem a necessidade de conhecimento técnico avançado. Em outras palavras, um CMS ajuda um usuário (ou mesmo uma pessoa desenvolvedora) a criar páginas Web sem necessariamente saber os detalhes do funcionamento do HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) e da linguagem de programação que o framework foi desenvolvido. Na prática, o CMS facilita o trabalho de alterar estilos e posicionamento de imagens na página, por exemplo. Quando deseja-se separar o gerenciamento e o armazenamento de conteúdo da apresentação desse conteúdo em sistema Web (**garcia2021**), utiliza-se um tipo de sistema de gerenciamento de conteúdo descentralizado denominado de *Headless* CMS[1] (detalhado na Seção 2.1.1). Em resumo, conforme a validação realizada neste trabalho, a estratégia White Label via um framework headless CMS mostrou-se adequada para os objetivos visando a escalabilidade e manutenibilidade do CPqs Abertas, em especial, por conta da proximidade da estrutura das páginas e da identidade visual dos sites integrantes da nova plataforma do projeto CPQs Abertas. Essa validação consistiu em migrar os sites já existentes (FAU, IME e FEARP) e inserir uma nova unidade (Instituto de Física - IF) no que podemos denominar de segunda geração da plataforma do CPQs Abertas, desenvolvida neste trabalho de conclusão de curso.

Este trabalho está estruturado em capítulos, além desta introdução, conforme a seguir. No Capítulo 2 é explicado os conceito, estratégias e tecnologias utilizadas para o desenvolvimento deste estudo e da nova plataforma proposta. O Capítulo 3 tem como enfoque

[1] https://aws.amazon.com/pt/what-is/headless-cms/

a arquitetura do projeto, de forma que é apresentada a implementação original do CPqs Abertas, comparando-a com o funcionamento da arquitetura proposta (em funcionamento atualmente) e as ações tomadas para solucionar os problemas que motivaram este trabalho. O Capítulo 4 descreve a implementação da nova plataforma, onde se analisa a metodologia de trabalho utilizada durante os meses de desenvolvimento, como a preparação e execução do projeto, estrutura do novo código, validação com a inserção de uma nova unidade, além de discutir o futuro do projeto, detalhando o que será necessário para a evolução do projeto. Por fim, a conclusão (Capítulo 5) aborda a importância do trabalho executado, destacando a migração do CPQs Abertas para uma nova arquitetura, baseada na estratégia White Label e headless CMS.

Capítulo 2

Conceitos e estratégias

Este capítulo concentra-se em apresentar o conceito por trás de um Sistema de Gerenciamento de Conteúdo (*Content Management Systems* - CMS) e a evolução recente desse modelo para o *Headless CMS*. Além disso, será explorado o uso específico do framework Strapi, um exemplo proeminente de Headless CMS, e sua relevância no desenvolvimento de soluções *White Label* para sistemas Web.

Além disso, serão examinadas as características, funcionalidades e vantagens dessas abordagens e tecnologias, bem como os benefícios e desafios de implementar a estratégia White Label utilizando um Headless CMS. Em termos práticos, discute-se como essa abordagem pode ser adotada, permitindo a personalização, escalabilidade e a entrega ágil de sites sob medida para diferentes contextos.

2.1 Sistemas de Gerenciamento de Conteúdo

Os Sistemas de Gerenciamento de Conteúdo (CMS) têm como objetivo centralizar e, assim, simplificar a criação, edição, organização e publicação de conteúdos de forma integrada em um sistema Web (**boiko2005content**). A produção colaborativa de conteúdo digital é o principal requisito atendido por um CMS.

O principal diferencial de um CMS é permitir ao usuário criar, publicar e editar o conteúdo de um site sem ter necessariamente um conhecimento de linguagens de programação ou de marcação (HTML). Isso deve-se à possibilidade de desenvolver um site apenas com as ferramentas disponibilizadas pelo CMS, com a possibilidade de escolhas como estilo para fontes, posicionamento dos elementos, inserção de imagens e diversas outras opções.

No geral, o núcleo de um CMS pode ser dividido em duas partes:

- **Content Management Application (CMA)**: permite a inserção dos elementos da página, como texto, imagens, etc.
- **Content Delivery Application (CDA)**: responsável pelos elementos a serem inseridos na página Web de forma responsiva ao usuário.

A abordagem em questão economiza tempo, esforço e custos de desenvolvimento significativos. Dessa forma, é possível direcionar os recursos com enfoque em aprimorar e personalizar as partes específicas de cada site. Outra vantagem é o padrão de qualidade, segurança e desempenho em todas as implementações de forma sólida e uniforme. Consequentemente, garante-se uma experiência consistente para os usuários e aumenta seus níveis de confiança e aprovação em relação à plataforma desenvolvida via um CMS.

No que diz respeito à manutenção, cria-se um cenário em que as atualizações de segurança, correções de bugs e melhorias de recursos podem ser aplicadas de forma centralizada, uma vez que todos os sites estão construídos sobre uma única base CMS compartilhada. Logo, todos os sites usufruem dos benefícios das últimas melhorias sem a necessidade de lidar com atualizações individuais. Além da economia de tempo, ocorre o fortalecimento da segurança e do desempenho geral, proporcionando uma gestão mais prática e eficiente da plataforma desenvolvida.

Por fim, a escalabilidade é uma das vantagens mais almejadas quando se utiliza uma base de CMS compartilhada para vários sites. Essa vantagem se deve, além da facilidade na expansão do número de sites sem a necessidade de recriar toda a infraestrutura, à rapidez no tempo de lançamento. É importante ressaltar que é possível manter a uniformidade na identidade visual e elementos-chave de design em todos os projetos. Ao compartilhar a mesma base, uma plataforma pode garantir uma consistência em todos os sites, mantendo logotipos, cores, tipografia e outros elementos de design alinhados. No entanto, essa flexibilidade tem limitações que podem ser contornadas ao também serem implementados módulos de personalização que permitam a adaptação de recursos específicos para atender às necessidades individuais de cada site, ao mesmo tempo que mantém a eficiência geral do CMS compartilhado. Dessa forma, pode-se equilibrar a rapidez no lançamento de novos sites com a capacidade de personalizar e atender às demandas exclusivas para um determinado site.

2.1.1 Headless CMS

Um *Headless CMS* separa o conteúdo da apresentação, fornecendo-o por meio de APIs (*Application Programming Interface*) para que os desenvolvedores possam escolher a forma que será apresentado. Conceitualmente, o termo "headless" refere-se à separação do front-end do back-end, de forma semelhante ao *Decoupled CMS*. Embora ambos apresentem vantagens em aplicações complexas, o *Headless CMS* se destaca ao enviar o conteúdo via API para diversos dispositivos, como dispositivos móveis e Internet das Coisas (IoT).

Ao comparar ambas as abordagens, a principal distinção está na flexibilidade de personalização. Para o contexto do CPqs Abertas, em que a principal preocupação diz respeito à possível limitação na personalização, o Headless CMS mostrou-se como a melhor alternativa. Enquanto o *Decoupled CMS* permite a desacoplagem do back-end e front-end, mas ainda mantém opções de visualização pré-construídas, o *Headless CMS* garante liberdade para criar interfaces personalizadas para diferentes plataformas. Em outras palavras, a abordagem escolhida oferece total controle sobre a Interface de Usuário (UI), o que permite a sua criação a partir do zero. Em contrapartida, essa maior flexibilidade de personalização implica em maior dificuldade no uso, implementação, manutenção e gerenciamento.

Ao estudar essa abordagem e conduzir uma prova de conceito usando o Strapi, um framework Headless CMS, a adoção dessa estratégia mostrou-se adequada para os objetivos pensados para o crescimento da plataforma do CPqs Abertas. Conceitualmente, a abordagem não se limita somente à separação do front-end e back-end, mas oferece uma vantagem crucial ao enviar o conteúdo através de APIs para diversos dispositivos. A comparação entre as opções consideradas destacou o fator definitivo relacionado à flexibilidade de personalização.

2.1.2 Strapi Framework

No atual cenário de desenvolvimento de sistema web, a gestão de conteúdo digital desempenha um papel fundamental na construção de sites e aplicativos interativos e dinâmicos. O Strapi destaca-se como uma solução inovadora nesse cenário que permite criar, modificar e organizar facilmente o conteúdo digital conforme as especificações do projeto. Além disso, é um software livre projetado para atender às necessidades de uma abordagem mais flexível na gestão de conteúdo, alcançada por meio de uma arquitetura de plugins extensíveis, que facilita a adição de funcionalidades personalizadas.

Uma das características que mais diferencia o Strapi é sua habilidade de gerar automaticamente APIs *RESTful* e *GraphQL* com base no conteúdo definido no sistema. Isso implica também na facilidade para uso em aplicativos web, móveis e outros sistemas. A capacidade de oferecer acesso a dados estruturados por meio de APIs é especialmente interessante. O Strapi foi concebido para poder ser integrado a vários tipos de bancos de dados. Isso possibilita aos desenvolvedores escolher o sistema de armazenamento de dados que melhor se alinha às necessidades de seus projetos, desde bancos de dados SQL tradicionais até soluções NoSQL.

Outra vantagem significativa deste framework é sua interface de usuário intuitiva e amigável, o que torna o Strapi acessível mesmo a usuários não técnicos. No entanto, a plataforma também oferece uma estrutura orientada para desenvolvedores que possibilita personalizações avançadas. Isso garante que as equipes de desenvolvimento possam criar experiências de usuário refinadas e complexas, ao mesmo tempo em que os editores de conteúdo consigam gerenciar o conteúdo de forma fácil, sem depender constantemente de intervenção técnica.

O núcleo do *Headless CMS*, presente no gerenciador de conteúdo do framework, é onde reside a essência do Strapi. Nele, são listados os chamados *Collection Types*, cuja definição depende do contexto específico do projeto. No caso do CPqs Abertas, temos *Institutos* e *Departamentos* como coleções distintas. Para ilustrar, *Departamentos* representam unidades específicas, por exemplo, Departamento de Ciência da Computação (DCC) do IME-USP, enquanto *Institutos* servem como representação das faculdades em si, como IME, FAU, FEARP. Cada *Collection Type* possui campos variados, como texto, JSON, números, relações, entre outros. O preenchimento desses campos é crucial, pois é por meio deles que a API é alimentada.

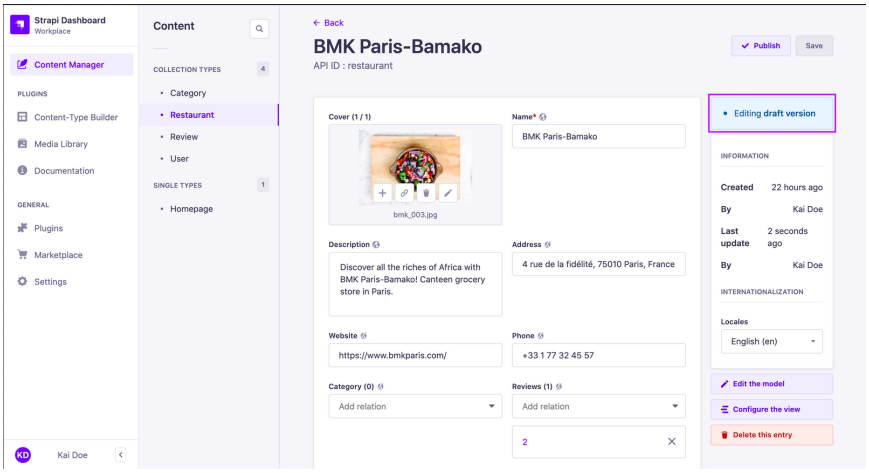


Figura 2.1: *Gerenciamento de conteúdo no Strapi: www.strapi.io*

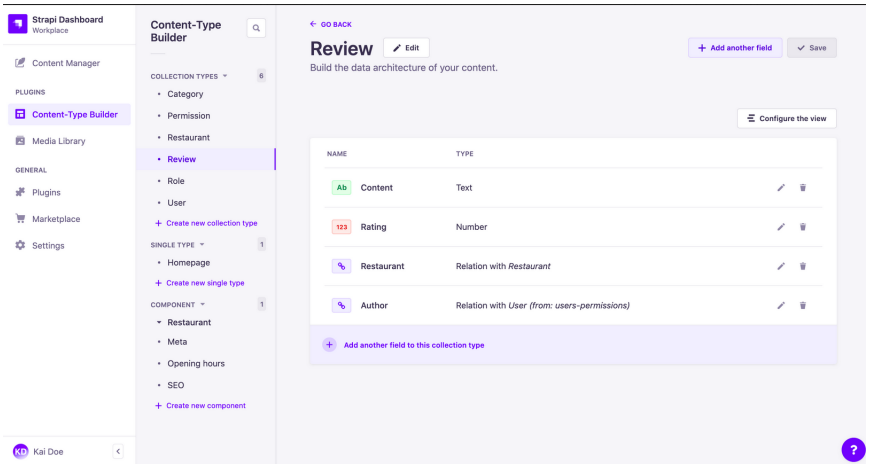


Figura 2.2: *Criação de conteúdo por tipo no Strapi: www.strapi.io*

As Figuras 2.1 e 2.2 mostram a interface padrão do Strapi. A primeira se refere ao gerenciamento de conteúdo, enquanto a segunda representa como é realizada a criação de cada campo desejado em determinada coleção.

Em relação à escolha tecnológica, as principais alternativas consideradas incluíram o Contentful e o GraphCMS, frameworks Headless CMS amplamente reconhecidos, porém são sistemas proprietários, já o Sanity foi a opção de software livre a ser considerada. Mas, por fim, decidiu-se utilizar a versão livre do Strapi (*Community Edition*), a qual está disponibilizada sob a *MIT Expat License* e possui um extenso ecossistema de plugins. Essa estratégia de abertura do código, mesmo com um licenciamento duplo, dado que o Strapi possui uma *Enterprise Edition*, amplia o conjunto de desenvolvedores capacitados a criar plugins, resultando em uma maior diversidade e quantidade de extensões disponíveis.

Embora a escolha pelo Strapi tenha apresentado inúmeras vantagens, algumas desvantagens foram identificadas, mas consideradas menos importantes para o caso do CPqs Abertas. Por exemplo, o GraphCMS e o Sanity oferecem recursos que aprimoram a colaboração em equipe, como controle de versão e atribuição de tarefas diretamente na plataforma.

No entanto, ao explorar o catálogo de plugins do Strapi, foram identificados plugins direcionados ao controle de versão (*Content Versioning*), enquanto a edição colaborativa poderia ser abordada por meio de um recurso específico, no caso, o Strapi Cloud, disponível apenas para usuários pagos.

Outra análise refere-se à manipulação de imagens e assets dinâmicos. Embora frameworks como Contentful e Sanity ofereçam abordagens diferenciadas para a manipulação desses dados, no caso do CPqs Abertas, a presença limitada de imagens, essencialmente relacionadas aos logotipos institucionais, minimizou a relevância desses recursos específicos.

Em resumo, o Strapi ganhou popularidade em diversos cenários de desenvolvimento web e de aplicativos, sendo frequentemente escolhido para projetos que demandam um alto grau de controle sobre o conteúdo e sua exposição por meio de APIs. Essa escolha abrange casos de uso, desde sites corporativos e blogs até aplicativos móveis, lojas online e sistemas de gerenciamento de conteúdo complexos. Essa versatilidade demonstra a adaptabilidade e a eficácia do Strapi em atender às necessidades de diferentes tipos de projetos.

2.2 White Label

White Label é um conceito de terceirização do desenvolvimento de serviços e produtos. Em termos práticos, envolve um serviço ou produto base que posteriormente será personalizado e redistribuído para outras empresas com o possível propósito de poupar custos e recursos de produção. A maior vantagem é flexibilizar a atribuição de responsabilidade do processo de criação e comercialização de um produto. Além disso, a definição não se restringe apenas a duas organizações. É possível, por exemplo, estabelecer três focos de responsabilidade – produção, marketing e vendas –, de forma que cada um seja atribuído a uma empresa distinta.

Historicamente, esse conceito tem raízes na indústria musical e na produção em massa comum a partir da década de 1950. Com o intuito de evitar gastos em registros não suficientemente lucrativos, as produtoras forneciam cópias "em branco" a DJs (Disk Jockey) reconhecidos que, por sua vez, as tocavam em eventos para analisar a recepção do público. Essa análise era decisiva na escolha dos registros que seriam lançados oficialmente. Atualmente, as principais áreas que adotam a prática são as indústrias de vestuário, de produtos alimentícios genéricos e de tecnologia financeira (FinTech).

Por exemplo, no contexto de produtos genéricos alimentícios, em diversos supermercados, é comum a presença de produtos personalizados com o logotipo da empresa onde são vendidos, denominados de "produtos de marca própria". Entretanto, esses estabelecimentos não são responsáveis pela manufatura do produto em questão. Outro exemplo, no contexto de equipamento de computadores, a empresa Dell contrata a produção de seus monitores, os quais são comercializados posteriormente sob sua marca. Ainda, como mais um exemplo, pensando em serviços, as centrais de atendimento são majoritariamente conhecidas pela adoção do White Label.

Para as empresas contratadas, a maior facilidade adquirida envolve a expansão da oferta visto que é possível contornar o impacto do ingresso no mercado no qual marcas

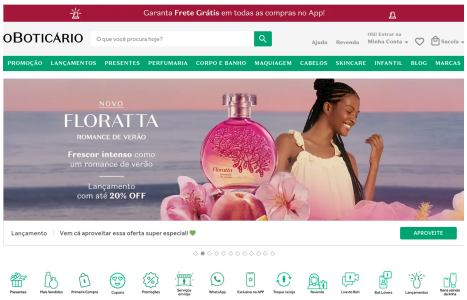
maiores e antigas exercem domínio. Em outras palavras, além de amenizar o risco de reprovação do produto, é possível fortalecer a presença da marca própria. Em contrapartida, a empresa proprietária poupa custos de produção e pode conduzir mais investimentos para o crescimento da empresa através do marketing e otimização de vendas. Dado que a solidificação das marcas deve-se à presença no mercado quantitativamente e ao alcance do público alvo, o White Label permite tanto a oferta de produtos em escala, como o enfoque na divulgação e crescimento da empresa.

No contexto de desenvolvimento de software, temos alguns exemplos bem estabelecidos da estratégia White Label:

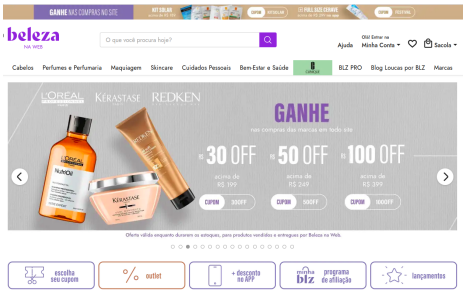
- **Shopify**: Uma empresa que desenvolve software para lojas online - nela é possível criar e gerenciar uma loja virtual. A ideia foi baseada em um software anterior desenvolvido pelos próprios fundadores da Shopify que servia como loja virtual de snowboard da qual eles eram proprietários.
- **Zendesk**: Empresa de desenvolvimento de software que oferece uma plataforma para o serviço de atendimento ao cliente através de tickets. Há também o oferecimento personalizado pela empresa, de forma que a empresa contratante pode ter o seu próprio atendimento por tickets.
- **Android**: Embora tenha o seu desenvolvimento liderado pela Google, é possível considerar o Android uma plataforma que se assemelha ao modelo *white label* no contexto dos smartphones. Essencialmente, esse sistema operacional apresenta flexibilidade, o que permite que outros fabricantes de smartphones o utilizem como base para seus dispositivos. Exemplos incluem a Samsung (One UI), a Xiaomi (MIUI) e a OnePlus (OxygenOS). Essas interfaces oferecem recursos exclusivos, layouts diferenciados e estilos visuais específicos, além de seus próprios aplicativos pré-instalados nos dispositivos.

No cenário da Tecnologia, os sistemas de software baseado em White Label, frequentemente, são disponibilizados sob o modelo Software as a Service (SaaS). O SaaS é um modelo de entrega de software baseado na nuvem, onde os aplicativos são acessados por meio da Internet (**turner2003turning**). A principal característica dessa abordagem é a flexibilidade e acessibilidade global para os usuários, com atualizações automáticas e a capacidade de acessar aplicativos de qualquer lugar. No contexto do White Label, o SaaS oferece às empresas a oportunidade de personalizar e comercializar serviços de software sob sua própria marca. Além disso, as empresas podem personalizar os recursos do aplicativo SaaS para atender a determinadas necessidades, o que envolve a adaptação de funcionalidades específicas e a adição de recursos exclusivos que são relevantes para seu mercado ou setor. Essa abordagem também permite que empresas se concentrem no atendimento às necessidades de seus clientes finais, fornecendo uma solução de software personalizada que atenda às demandas do mercado, em vez de gastar recursos no desenvolvimento inicial.

Dessa forma, empresas de tecnologia muitas vezes disponibilizam seus sistemas baseados na estratégia *White Label*. Por exemplo, as Figuras 2.3(a) e 2.3(b) mostram as páginas iniciais das plataformas do “O Boticário” e “Beleza na Web”, respectivamente. Com base neste exemplo foi tomada a decisão sobre o uso da estratégia White Label para a evolução



(a) O Boticário: www.boticario.com.br

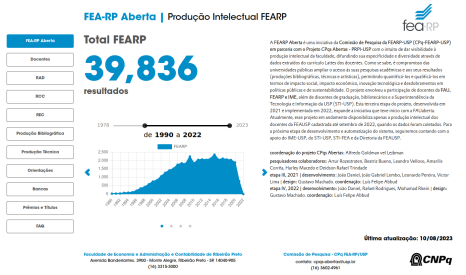


(b) Beleza na Web: www.belezanaweb.com.br

Figura 2.3: *Página inicial de dois sites baseados em White Label*

da plataforma CPqs Abertas.

(a) Instituto de Matemática e Estatística



(b) Faculdade de Economia e Administração e Contabilidade de Ribeirão Preto

Figura 2.4: *Plataformas distintas do projeto*

As Figuras 2.4(a) e 2.4(b) mostram as semelhanças dos sites de dois institutos (IME e FEARP) na plataforma CPqs Abertas, de maneira similar ao exemplo apresentado anteriormente. Em suma, observa-se a manutenção dos elementos de identidade visual do projeto e, simultaneamente, permitir a personalização de cada plataforma.

Este capítulo teve como enfoque explicar ferramentas e estratégias utilizadas para o desenvolvimento do projeto, dando exemplos de aplicações reais da estratégia *White Label*, “ O Boticário” e “Beleza na web”, e ferramentas CMS extensamente utilizadas, como o Sanity e o próprio Strapi. O próximo capítulo visa detalhar a arquitetura do CPQs Abertas investigando a anterior, explicando sua organização e funcionamento, além de discutir a nova arquitetura e os desafios encontrados para a migração.

Capítulo 3

Arquitetura

Arquitetura refere-se à estrutura fundamental de um sistema e à maneira como seus componentes interagem. Ela envolve a organização e o design dos diversos elementos que compõem um software, como módulos, componentes e serviços, assim como as relações entre eles. A arquitetura de software é crucial para garantir que um sistema seja eficiente, escalável, fácil de manter e cumpra seus requisitos funcionais e não funcionais (**engsoftmoderna**).

A nova arquitetura do CPQs Aberta foi planejada para superar os desafios pré-existentes do projeto. Anteriormente, fragmentada em instâncias individuais, essa estrutura arquitetônica prévia gerava limitações que resultaram em dificuldades operacionais e custos excessivos. A compreensão desses obstáculos guiou a transição para a arquitetura atual, com o objetivo de proporcionar uma infraestrutura mais escalável e adaptável ao projeto.

3.1 Arquitetura Antiga

A arquitetura estava dividida em quatro partes:

- Banco de dados PostgreSQL, um sistema de gerenciamento de dados relacionais;
- Banco de dados MongoDB, um sistema de gerenciamento de banco de dados NoSQL de código aberto, orientado a documentos;
- O back-end em Django, um framework para web escrito em Python, que utiliza o padrão model-template-view;
- Front-end em React, uma biblioteca de JavaScript que é utilizada para desenvolvimento web;

Todas as partes estavam agrupadas em um mesmo repositório; em termos de organização de diretórios, todas essas camadas do projeto estavam no mesmo local. Ou seja, o back-end e o front-end estavam no mesmo diretório, o que agravava ainda mais a estrutura do projeto como um todo, pois gerava confusão no entendimentos das pastas e arquivos. Cada site (instituto) possuía seu código isolado e uma instância dedicada para o banco de dados. Além disso, eram criados containers (unidades de software leves e autossuficientes

que incluem o necessário para executar um aplicativo, como código, bibliotecas, dependências e configurações do sistema) no Docker (uma plataforma de software livre que facilita a criação, empacotamento e distribuição de aplicativos em ambientes isolados), para cada camada da arquitetura. Isso resultava em quatro *containers* distintos: um para o front-end, outro para o back-end e outros dois para os bancos de dados.

Esses containers eram então instanciados no serviço de nuvem da AWS (*Amazon Web Services*) chamado EC2 (*Amazon Elastic Compute Cloud*), que cria máquinas virtuais para a aplicação. Assim, uma única VM (*Virtual Machine*) era responsável por conter o back-end, front-end e banco de dados de um site específico. Do ponto de vista técnico, o sistema anterior era estruturado em uma arquitetura rígida, onde o front-end e o back-end estavam vinculados em um único código-fonte.

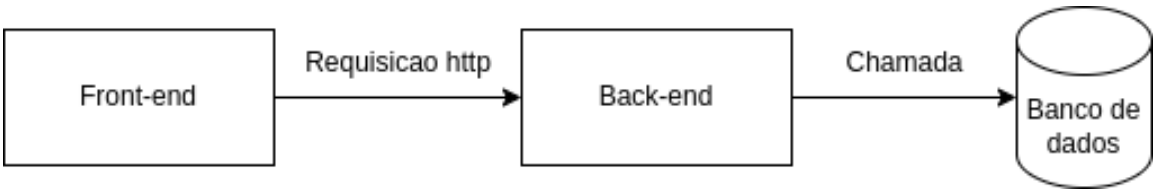


Figura 3.1: *Estrutura geral antiga*

A Figura 3.1 ilustra, de forma geral, a estrutura na prática do sistema para um determinado instituto. Considerando o modelo de arquitetura apresentado na Figura 3.1, o procedimento para inserir um novo instituto exigia a replicação integral do código: front-end, back-end e a criação de um novo banco de dados independente. Isso resultava na criação separada de cada front-end, originando pequenas particularidades no desenvolvimento, embora os códigos-fontes seguissem um modelo com considerável replicação entre os sites. No back-end, a cada novo instituto, era necessário criar uma nova instância com código idêntico aos anteriores. O mesmo padrão era aplicado ao banco de dados, ou seja, uma nova instância era criada para cada instituto adicional.

Com esse cenário, baseado em um estrutura de código replicado, a manutenção do projeto tornou-se progressivamente mais complexa, trabalhosa e propensa a erros, uma vez que cada alteração demandava modificações em cada instância de código individualmente. Por exemplo, ao inserir um novo botão em todos os sites, era necessário modificar cada código-fonte de cada instância separadamente, resultando em desafios na manutenção dos sites, em especial ao longo do tempo com a adição de mais institutos no CPqs Abertas.

Com a arquitetura anterior, foram desenvolvidos sites para três institutos distintos: Faculdade de Arquitetura e Urbanismo (FAU), Instituto de Matemática e Estatística (IME) e Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto (FEARP), o que demandou o uso de três instâncias de máquinas virtuais na AWS. Por exemplo, essa abordagem resultou em um aumento de custos, uma vez que era necessário criar uma nova máquina virtual toda vez que um novo instituto era inserido no projeto. Embora, de acordo com a lógica de escalabilidade horizontal, isso fosse considerado uma qualidade do projeto, percebeu-se que a demanda pelo back-end era relativamente pequena. Portanto, concluiu-se que não seria necessário alocar uma máquina virtual exclusiva para a execução do back-end. A disposição original das máquinas virtuais na AWS pode ser visualizada na Figura 3.2.

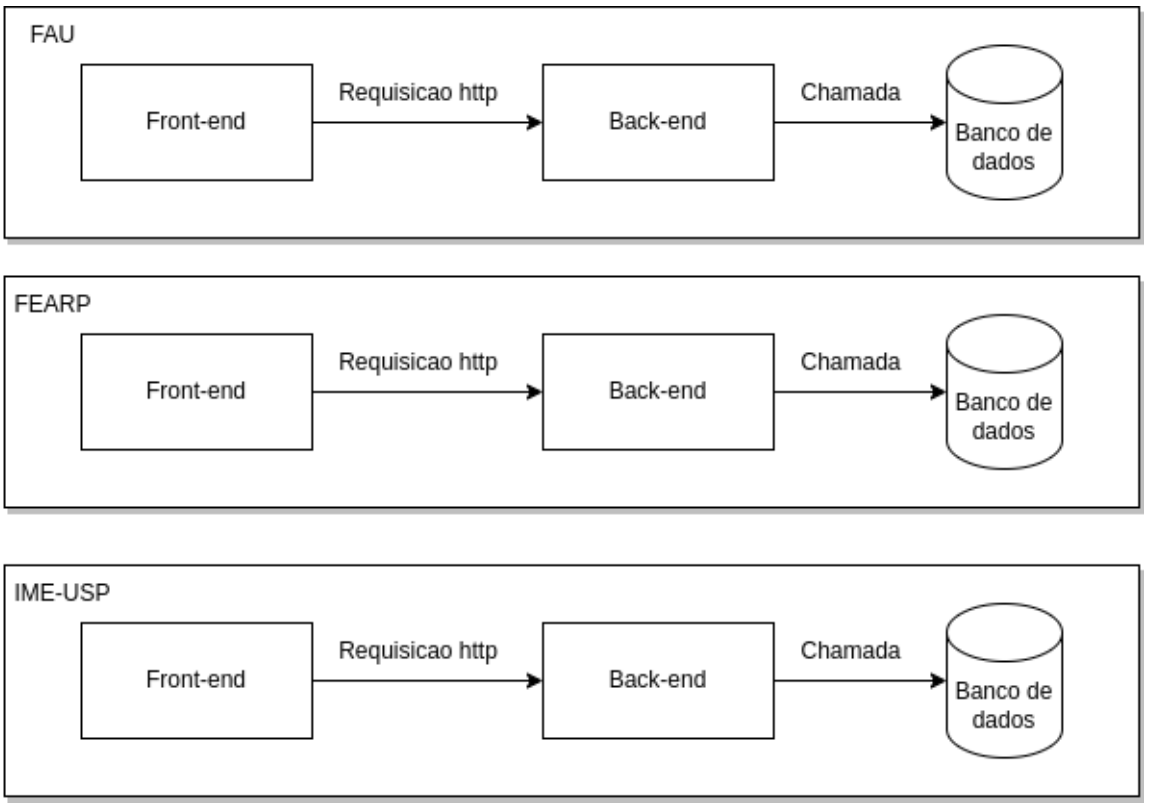


Figura 3.2: *Estrutura antiga com os institutos que foram criados*

Outro problema presente na abordagem da arquitetura original é a questão da segurança. A manutenção de todos os componentes agrupados em um único código base implica o compartilhamento do mesmo contexto de segurança. Em outras palavras, isso resulta em uma superfície de ataque maior, uma vez que uma vulnerabilidade em uma parte do sistema pode comprometer toda a plataforma.

Em resumo, a própria arquitetura se tornava um obstáculo para a escalabilidade do projeto, além de apresentar as dificuldades relacionadas à obtenção de dados. Os custos associados à inclusão de novas unidades envolviam despesas financeiras com mais máquinas virtuais, trabalho desnecessário por parte dos desenvolvedores e tempo a ser dedicado.

3.2 Arquitetura Atual

Visando corrigir e mitigar os problemas descritos quanto às decisões arquiteturais que estruturaram o projeto CPqs Abertas original, neste trabalho, buscou-se criar uma nova arquitetura para desacoplar os diferentes componentes do projeto. Inicialmente, separou-se o back-end e o front-end em diferentes repositórios para aprimorar a organização do projeto. A arquitetura *Headless*, que divide as responsabilidades entre o front-end, responsável pela interface de usuário, e o back-end, responsável pelo CMS e pela lógica de negócios, foi identificada como o estilo mais benéfico ao analisar os objetivos do projeto CPqs Abertas.

Anteriormente, cada instituto possuía um banco de dados próprio gerado dentro de um container Docker. Agora, esses bancos de dados individuais foram consolidados em uma única estrutura, utilizando uma tecnologia chamada RDS (*Relational Database Service*) da Amazon, que armazena banco de dados PostgreSQL. Essa nova estrutura de bancos de dados agregados possibilitou a unificação do back-end, uma vez que as requisições dos sites podiam ser filtradas, permitindo direcionar o back-end para o banco de dados apropriado em função dos diferentes tipos de consultas.

Os diferentes componentes de back-end e bancos de dados criados para cada instituto foram unificados. Isso significa que os sites dos institutos poderiam requisitar dados de uma única API, resultando em apenas uma VM na AWS executando o back-end. Essa mudança não apenas reduziu os custos, mas também simplificou o processo de manutenção. Uma nova camada para o CMS foi criada utilizando o Strapi, sendo responsável por gerir os estilos e estados dos diferentes sites. Nela, foi possível armazenar os estilos de cada site e modificá-los em tempo de execução. Em outras palavras, possibilitou a unificação do código-fonte do front-end, tornando viável a criação de páginas distintas utilizando a mesma estrutura. Essa nova arquitetura proporciona a facilidade na inserção de novos institutos, além de contribuir para a manutenção dos já existentes.

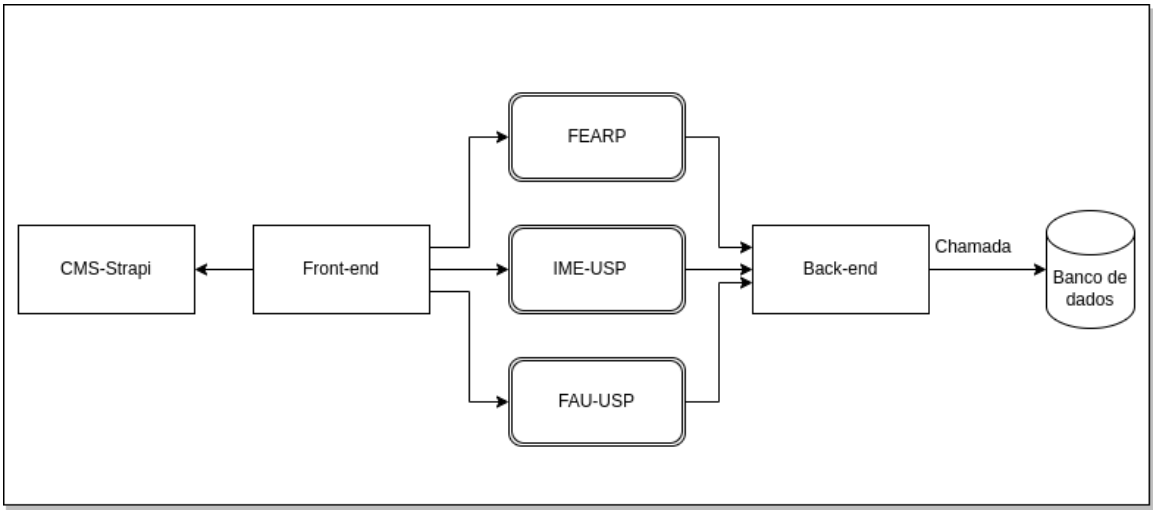


Figura 3.3: *Estrutura novas com os institutos que foram criados*

A Figura 3.3 representa essa nova estrutura com os institutos já existentes no CPqs Abertas. Pela análise das Figuras 3.2 e 3.3, é possível compreender melhor a questão da unificação. Na nova abordagem, há um front-end que realiza uma requisição para o CMS, gerando diferentes estilos para os institutos. Quando o site é gerado, ocorre uma requisição para o back-end obter os dados referentes ao instituto em questão, direcionando a API para o banco de dados apropriado.

Atualmente, a instância do front-end utiliza o serviço da Amazon S3 (*Simple Storage Service*), que armazena arquivos e hospeda sites estáticos, como é o caso deste projeto que utiliza ReactJS. Para a hospedagem, utiliza-se a funcionalidade de *build* do framework, que gera um código HTML e, em seguida, é enviado para o serviço de armazenamento da Amazon. Dessa forma, não é mais necessário utilizar um container e uma VM para executar o front-end.

```
REACT_APP_INSTITUTE_CMS=FAUUSP
REACT_APP_INSTITUTE=fau
```

Figura 3.4: *Variáveis de ambiente utilizadas para gerar os sites*

Na arquitetura atual, a construção do código ocorre de forma unificada para todos os sites, sendo a única distinção a especificação na variável de ambiente, como exemplificado na Figura 3.4, indicando a qual instituto o processo está associado. Dessa forma, no novo esquema, o layout do site é gerado dinamicamente durante a execução. Em outras palavras, os estilos do front-end são gerados assim que o usuário decide utilizar a página desejada.

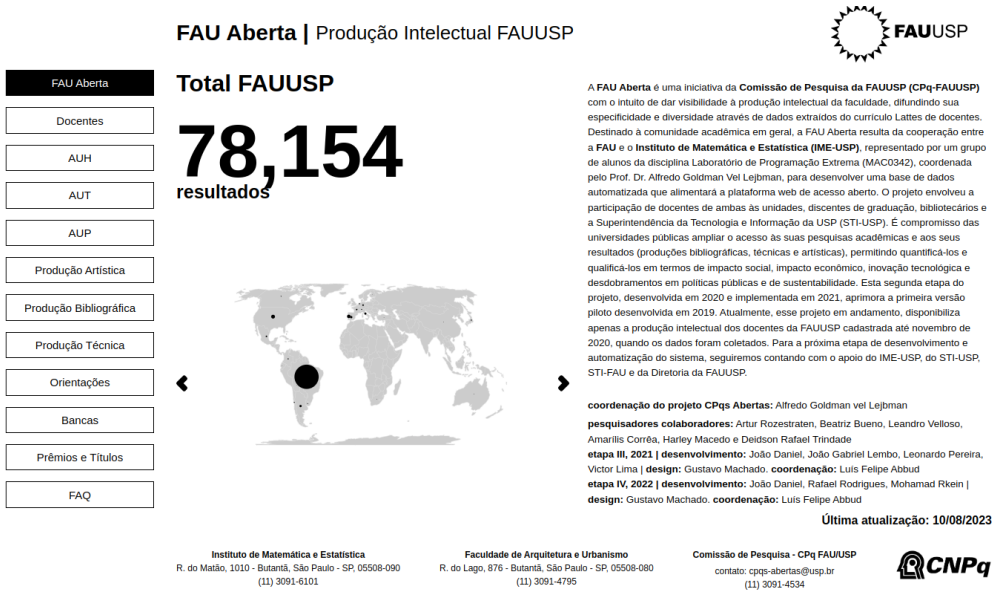


Figura 3.5: *Variáveis ambientes com valores para FAU*

Na Figura 3.4, a variável REACT_APP_INSTITUTE indica qual banco de dados será utilizado nas requisições para o back-end, enquanto a variável REACT_APP_CMS representa o instituto utilizado pelo CMS para gerar os estilos. Nesse exemplo, o site da FAU é criado (Figura 3.5), juntamente com os dados específicos associados a ela. Com isso, ao alterar os valores dessas variáveis para os correspondentes ao IME, o site é criado com os estilos do novo instituto, utilizando o mesmo código-fonte, o que mostra a facilidade da formação de novos estilos e principalmente na construção para novos institutos, conforme demonstrado na Figura 3.6.

Em síntese, a transição da arquitetura original para a atual representa não apenas uma adaptação técnica, mas uma resposta estratégica aos desafios operacionais e limitações identificadas para a evolução do projeto CPqs Abertas. A mudança para uma arquitetura desacoplada, não apenas otimizou a eficiência e escalabilidade do sistema, proporcionou uma redução significativa nos custos operacionais. Ao consolidar os bancos de dados, unificar os componentes de back-end e introduzir ferramentas como o Strapi, a nova

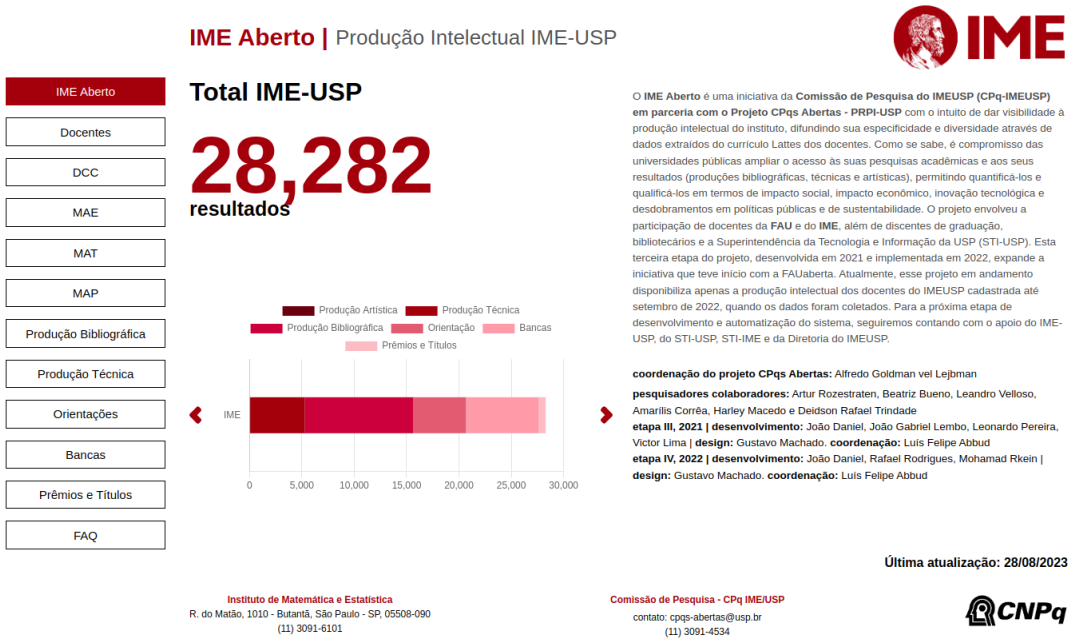


Figura 3.6: *Variáveis ambientes com valores para o IME*

arquitetura oferece uma base sólida para a inserção de novos sites (institutos) de forma mais simplificada. Além disso, a hospedagem mais eficiente com o Amazon S3 eliminou a necessidade de recursos computacionais excessivos. Essas melhorias não apenas superou as dificuldades pré-existentes, mas também estabeleceu um alicerce robusto, preparando o projeto para melhor lidar com possíveis outros desafios no futuro.

Capítulo 4

Implementação

O capítulo anterior delineou os desafios e limitações enfrentados pela estrutura anterior, destacando as dificuldades inerentes à replicação do código, à escalabilidade, aos custos e às questões de segurança. A necessidade de uma mudança tornou-se evidente, impulsionando a equipe de desenvolvimento a buscar uma abordagem mais flexível e eficaz.

Com a visão clara dos problemas enfrentados na arquitetura legada, este capítulo se concentra na implementação prática das soluções propostas. A nova arquitetura foi projetada e implementada, adotando uma abordagem *Headless* que separa o front-end e o back-end, ou seja, descentralizada, além de consolidar os bancos de dados em uma estrutura unificada.

O processo de transição não apenas implicou mudanças estruturais, mas também exigiu uma redefinição dos fluxos de trabalho, integração de novas tecnologias e a reorganização dos componentes centrais do sistema. A busca por uma arquitetura mais escalável, modular e economicamente viável levou a uma série de escolhas técnicas e estratégias de implementação que serão exploradas detalhadamente neste capítulo.

Ao adotar uma visão mais granular do sistema, buscando desacoplar suas partes e unificar funcionalidades comuns, a nova arquitetura representa não apenas uma mudança estrutural, mas uma transformação na forma como os sites (institutos) são integrados, gerenciados e mantidos no CPqs Abertas.

Este capítulo trata da implementação da nova arquitetura, detalhando as escolhas tecnológicas, os desafios enfrentados durante o processo de transição e os benefícios obtidos com a abordagem adotada. A análise das etapas de implementação revelará não apenas as soluções aplicadas, mas também o impacto dessas mudanças na operacionalidade, manutenção e escalabilidade do sistema.

Como resultado da nova implementação, os seguintes sites encontram-se implementados na nova arquitetura:

- Faculdade de Arquitetura e Urbanismo (FAU): http://cpqs-abertas-fau.s3-website-sa-east-1.amazonaws.com/
- Instituto de Matemática e Estatística (IME): http://cpqs-abertas-ime.s3-website-sa-east-1.

amazonaws.com/

- Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto (FEA-RP): http://cpqs-abertas-fearp.s3-website-sa-east-1.amazonaws.com/
- Instituto de Física (IF): http://cpqs-abertas-fisica.s3-website-sa-east-1.amazonaws.com/
- Hub do CPqs Abertas: http://hub-cpqs-abertas.s3-website-sa-east-1.amazonaws.com/

Além desses, a administração do CMS Strapi e o repositório com o código-fonte do projeto estão disponíveis, respectivamente, em: http://ec2-3-87-224-44.compute-1.amazonaws.com/ e https://gitlab.com/cpqs-abertas/cpqs-abertas-v2

4.1 Organização e gestão das tarefas

A nova plataforma do CPqs Abertas é um software livre, sob a licença MIT (*Massachusetts Institute of Technology*)[1], disponibilizado no GitLab[2]. Sendo um software livre, durante o desenvolvimento, deve-se manter uma documentação mínima e fornecer um canal de comunicação aberto para os possíveis colaboradores. No caso do CPqs Abertas, a cada ciclo de desenvolvimento, a documentação foi obrigatoriamente atualizada para incluir explicações sobre instalação, configuração e uso do software. Isso facilita a contribuição de novos desenvolvedores para o projeto. No que diz respeito ao canal de comunicação, as *issues* são o principal meio de interação, de forma que aquelas relacionadas às tarefas de um ciclo eram atualizadas conforme sua execução. Adicionalmente, foi organizado um procedimento para adicionar e documentar cada uma das issues pendentes, visando facilitar para os próximos desenvolvedores.

Durante o desenvolvimento, a equipe (autores deste trabalho) adotou práticas das metodologias ágeis. A abordagem ágil foi escolhida devido aos benefícios de sua natureza adaptativa e personalizada, proporcionando não apenas desenvolvimento mais rápido e eficaz, mas também maturidade à organização da equipe, conforme discutido por **8229928**.

Nesse contexto, os principais marcos temporais do ciclo de desenvolvimento foram estabelecidos como *Milestones* no repositório. Em resumo, esses marcos representam metas específicas a serem alcançadas em momentos determinados, auxiliando na divisão do processo em fases gerenciáveis. Isso permite o acompanhamento do progresso e a avaliação do cumprimento de objetivos em momentos-chave.

Além disso, dentro de cada Milestone, a administração era conduzida por meio da visualização em forma de *Kanban*, conforme ilustrado na Figura 4.1. Essa ferramenta visual, geralmente, consiste em colunas representando etapas como “To Do”, “In Progress” e “Done”. Dessa forma, foi possível obter uma visão rápida do status de cada issue/tarefa.

[1] https://opensource.org/license/mit/

[2] https://gitlab.com/cpqs-abertas/cpqs-abertas-v2

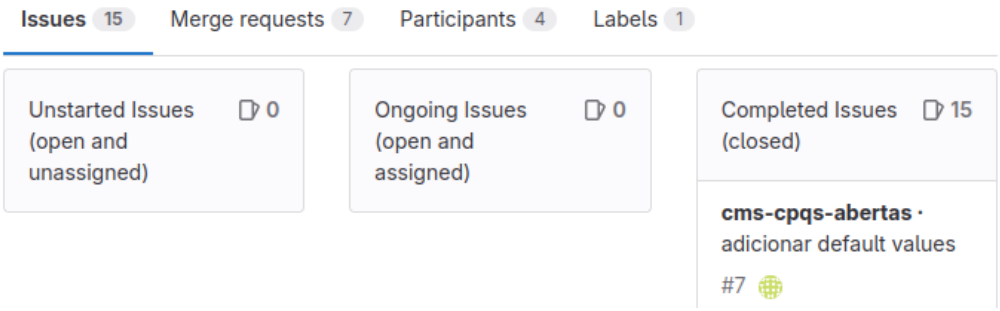


Figura 4.1: *Milestone: visualização Kanban*

As *issues* (ou tarefas), por sua vez, podem se referir a bugs, melhorias ou qualquer unidade de trabalho identificada e registrada no sistema de controle e gerenciamento do projeto. É importante ressaltar que não havia um formato padrão de issue: algumas eram abrangentes e continham subitens para auxiliar no entendimento, enquanto outras eram curtas e gerais.

Para cada issue realizada, o código referente à melhoria era adicionado a uma *branch*, ou seja, a uma versão independente do código-fonte no repositório do Gitlab. Esse uso permitiu que todos os desenvolvedores da equipe trabalhassem em recursos separados sem interferir uns nos outros.

Ao concluir a issue, era efetuado um *Pull Request*, ou seja, uma solicitação para incorporar as alterações de uma branch ao código principal do projeto (*branch main*). A aprovação dessas requisições ocorria após a realização da revisão de código, prática essencial em metodologias ágeis. Essa estratégia envolve a revisão sistemática e colaborativa do código por outros membros da equipe. Durante cada revisão, o membro responsável garantia que o código estivesse em conformidade com os padrões de codificação, diretrizes e práticas estabelecidas pela equipe.

4.2 Preparação

Além da estratégia de separação em ciclos, foi necessário elaborar um plano mais definido para a implementação, o qual incluísse objetivos, cronograma, recursos necessários e etapas pré-definidas de execução. Na análise para o CPqs Aberta, permitir a personalização completa da identidade visual e do conteúdo para os diferentes clientes da plataforma (os institutos) era essencial. Além disso, era fundamental garantir o isolamento de dados e acesso, ou seja, garantir que, embora a interface compartilhasse o mesmo código-fonte, cada fluxo de dados fosse entregue exclusivamente ao seu front-end correspondente. No que diz respeito ao acesso, era necessário garantir que os clientes intermediários, que teriam acesso ao Strapi, não tivessem acesso ao código-fonte e outras ferramentas de responsabilidade exclusiva dos desenvolvedores. Por fim, seria necessário avaliar a facilidade de administração da nova estrutura de gerenciamento de plataformas, mantendo-a intuitiva para os designers, sem comprometer a eficiência.

Para assegurar a entrega dentro do prazo pré-estabelecido de oito meses, considerando a complexidade e magnitude do projeto, foi necessário utilizar a estratégia *"divide*

after you conquer" dentro da metodologia ágil (**elshamy2006divide**). Assim, para cada etapa definida, foram estabelecidos prazos médios que considerassem a complexidade das personalizações, integrações e estudos necessários.

Inicialmente, era essencial familiarizar-se com o código legado, uma etapa fundamental em qualquer projeto desse tipo. Isso não significava apenas ler o código-fonte, mas executá-lo na própria máquina, realizar mudanças para compreender o funcionamento de cada parte do código. Essa etapa era crucial para qualquer subsequente, agilizando o processo de implementação. Embora a equipe de desenvolvimento tenha estimado um período de um mês, foram necessários dois meses para compreender totalmente o código-fonte.

Em seguida, era necessário estudar as opções de sistemas de software no mercado que atendessem ao nosso objetivo principal. Isso envolveu reuniões com profissionais da área para identificar a estratégia mais adequada ao projeto. Após essa definição, o próximo passo foi decidir quais tecnologias seriam utilizadas no projeto e como seriam implementadas. Essa etapa teve a duração esperada de um mês, conforme estipulado no início do cronograma.

O ciclo prático, referente à implementação efetiva, teve a maior duração entre as etapas, considerando os possíveis obstáculos durante o desenvolvimento. Esse processo durou aproximadamente três meses, principalmente devido a inconsistências encontradas ao incorporar a nova arquitetura, que precisaram ser contornadas.

Os dois últimos meses foram dedicados à etapa de validação da nova arquitetura, incorporando um novo instituto ao CPqs Abertas (no caso, o Instituto de Física). Depois de todos os ajustes na nova plataforma, a partir do momento em que a equipe teve acesso aos dados necessários, levou-se menos de duas horas para acoplar a nova unidade ao sistema CPqs Abertas, incluindo o tempo necessário para definir a identidade visual.

Outro fator essencial no cronograma eram as revisões periódicas, realizadas a cada duas semanas aproximadamente, em conjunto com o orientador do projeto, garantindo o alinhamento com os objetivos iniciais e os prazos estabelecidos.

Além disso, uma avaliação dos recursos foi necessária. Em relação aos recursos humanos, foi fundamental identificar a equipe além dos desenvolvedores que garantiria o maior sucesso em cada etapa, incluindo designers após a estabilização do nosso sistema, para contribuir com conceitos de interface e experiência do usuário (UI/UX). Essa equipe incluiu um consultor líder e dois estudantes de design da Faculdade de Arquitetura e Urbanismo da USP.

Também era essencial avaliar a estrutura tecnológica existente para garantir a disponibilidade dos recursos de hardware, software e servidores necessários para hospedar a nova estrutura. Nesse contexto, foram realizadas as mudanças detalhadas no Capítulo 3, reduzindo os custos com servidores/máquinas virtuais associados à introdução de novas unidades ao CPqs Abertas.

Vale ressaltar que as avaliações periódicas do projeto permitiram ajustar estratégias conforme necessário, seguindo a metodologia ágil. Um exemplo foi a mudança na estrutura visual do projeto. Inicialmente, realizar uma mudança estrutural desse porte não era prioridade, porém, a partir do olhar técnico da equipe de designers, foi possível estipular

para um etapa futura, fora do escopo deste trabalho, modificações para garantir a melhor experiência do usuário.

4.3 Execução

Para a implementação do que foi planejado para o novo CPqs Abertas, foi necessária compreensão da situação na qual se encontrava o projeto original, estudando código legado, compreendendo como era feita a execução e também *frameworks* utilizados. Nesse processo de estudo, notou-se que o projeto foi desenvolvido utilizando *React*, biblioteca front-end do *JavaScript*. Sabendo disso, foi necessário uma fase de adaptação por alguns membros, uma vez que, nem todos tinham experiência com essa biblioteca.

Devido à complexidade do código legado e o estado de organização em que se encontrava o projeto, somente depois de algumas semanas de estudos houve a segurança necessária para iniciar a implementação da nova versão. O marco inicial foi a definição pela estratégia *White Label* e a indicação por parte de um especialista para o uso do do Strapi como o CMS para o CPqs Abertas.

Com isso, foi feito o POC (*Proof of Concept*) para checar se a estratégia faria sentido ao projeto. Após essa validação, começaram os estudos da nova ferramenta, o Strapi, onde atingiu-se a compreensão das funcionalidades oferecidas pela ferramenta, bem como sua utilização. Em paralelo a isso, começou-se a trabalhar em uma falha no projeto previamente detectada, a sua falta de modularização.

Finalmente, com o projeto devidamente organizado e modularizado, a conversão do sistema para um CMS iniciou-se. Nesta etapa maior, também ocorreram duas tarefas em paralelo: a adaptação do front-end, para comportar a nova estrutura em CMS, e do back-end, em que para cada instituto houve uma reprodução no método de extração de dados para que o novo banco de dados fosse gerado. Além disso, as devidas adaptações foram realizadas para o uso de apenas um banco unificado. Especificamente, entre as adaptações realizadas do front-end, destaca-se a organização e uso do Strapi, o plugin CKEditor, a aplicação do contexto, as adaptações no CSS e as mudanças na geração de gráficos.

Organização e uso do Strapi. Na página responsável pelo gerenciamento de conteúdo fornecida pelo Strapi, é importante destacar o uso de duas seções:

- Content Type Builder: na qual são criadas as chamadas *Collection Types*, *Single Types* e *Components*. Em outras palavras, responsável pela criação dos campos de cada instância e componente de instância.
- Content Manager: na qual ocorre o preenchimento dos campos criados, ou seja, são os dados que caracterizam cada instância.

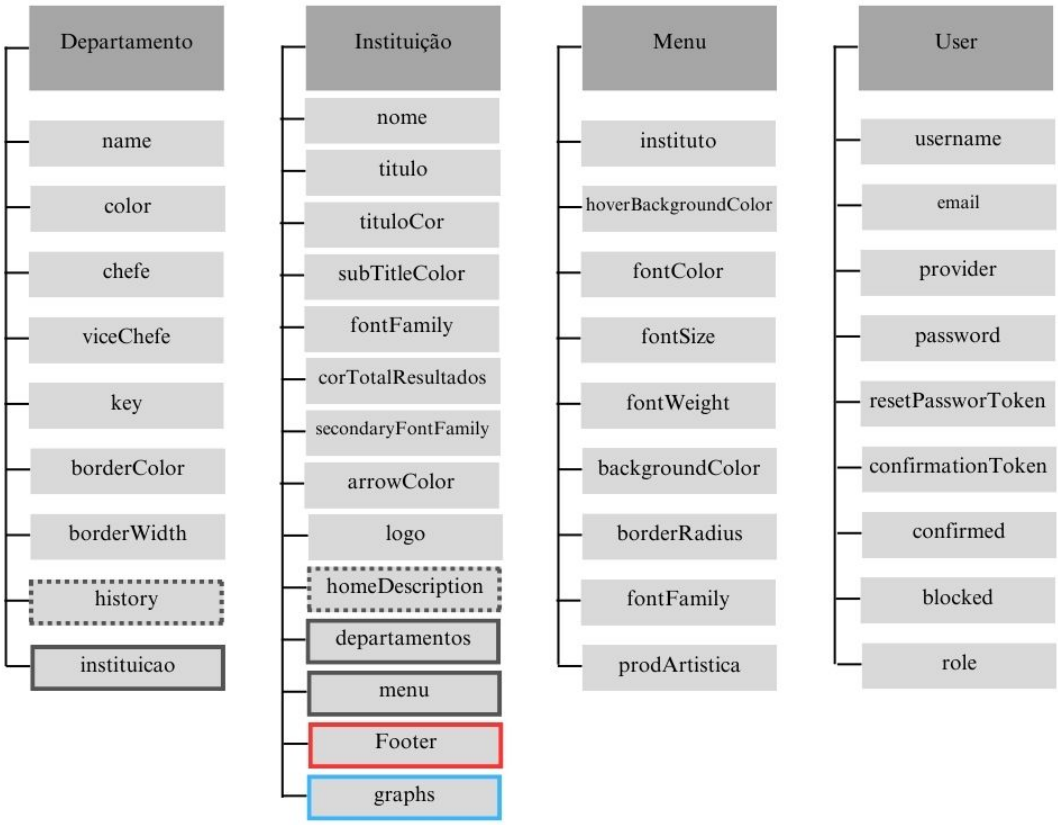


Figura 4.2: *Organização: Collection Types*

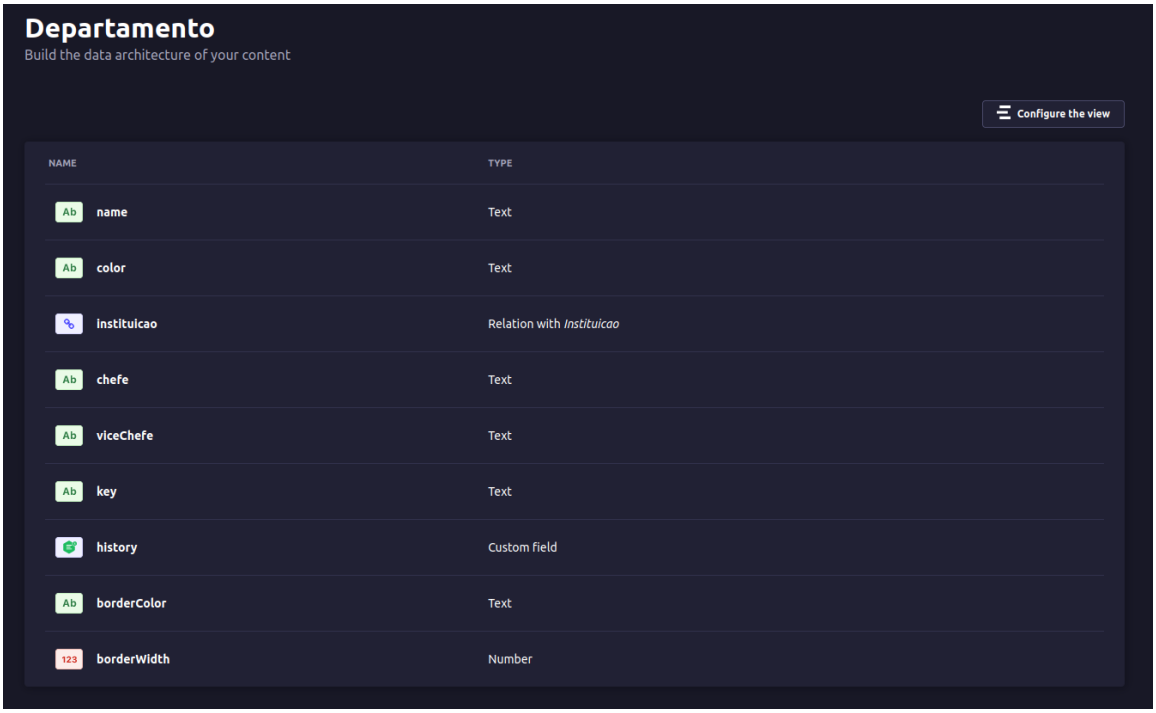


Figura 4.3: *Implementação de um Collection Type*

A organização geral do projeto na plataforma pode ser definida por quatro coleções principais, das quais cada uma possui seus próprios campos, como pode ser visto na Figura 4.2 (note que a figura exemplifica apenas uma coleção). Para os campos, o Strapi oferece tipos fixos, dos quais foram utilizados *Text*, *Number*, *Boolean*, *Enumeration* e *JSON*; além de campos do tipo relação, marcados na Figura 4.2 com borda contínua. A interface da construção das coleções com seus campos e respectivos tipos está representada na Figura 4.3.

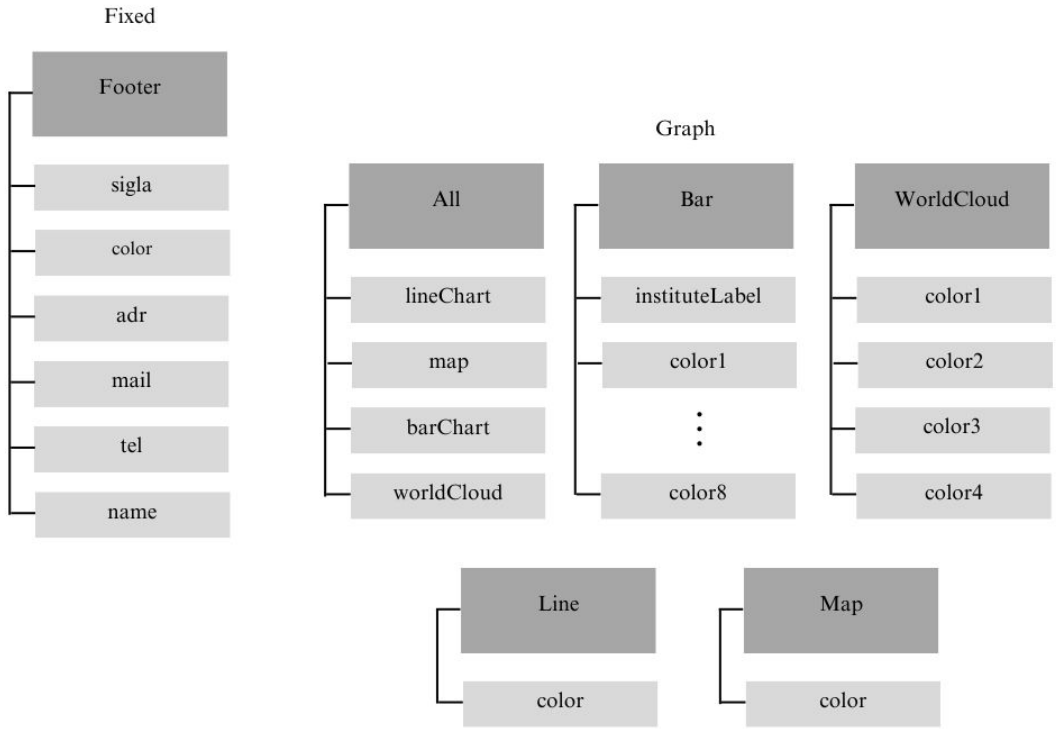


Figura 4.4: *Organização: Components*

Em adição aos elementos de interface, existe uma coleção *User*, responsável pelo controle de usuários que utilizam a plataforma. Isto é, armazena os dados que garantem os direitos de edição e/ou visualização, além de controlar o fluxo de *login*. Os campos relação oferecem tipagem variada, além de sentido único ou múltiplo. Na nova plataforma do CPqs Abertas, foram utilizadas as relações de sentido único, no qual cada *Instituição* possui um *Menu*, e de um-para-muitos, entre *Departamento* e *Instituição*. Outro campo oferecido importante é o tipo componente, que constituem estruturas reusáveis e compartilháveis entre qualquer *content type*, incluindo outros componentes. Os componentes implementados no CPqs Abertas podem ser vistos na Figura 4.4, e constituem, sobretudo, o gerenciamento dos estilos dos gráficos.

No mesmo contexto, temos as chamadas *Dynamic zones* (zonas dinâmicas), que se tratam de uma combinação de componentes, cujas estruturas podem variar. Em outras

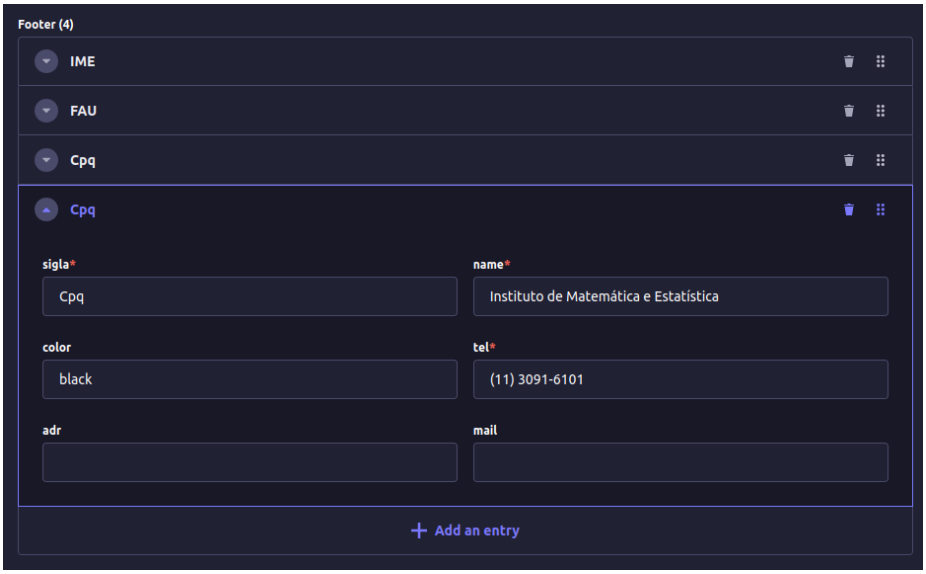


Figura 4.5: *Componente com repetição em Footer.*

palavras, trata-se de um campo no qual é possível adicionar qualquer componente da lista pré-selecionada, que, por sua vez, tem seus próprios campos definidos. Essas zonas dinâmicas podem ser úteis para a personalização diversa e única de determinados institutos, visto que permitem a inclusão opcional de determinados componentes. Embora não utilizadas nessa primeira implementação, outro campo de característica personalizável utilizado foi o componente passível de repetição, para o *Footer*. Seu funcionamento ocorre como ilustrado na Figura 4.5, ou seja, é possível adicionar inúmeras entradas, que serão reorganizadas na plataforma proporcionalmente, no caso, lado a lado no rodapé.

Por fim, o Strapi oferece a possibilidade de definir valores *default*, que serão automaticamente preenchidos na criação da instância. Além disso, também permite que, no processo de construção de um campo, sejam definidas expressões regulares para garantir seu preenchimento correto e o caráter obrigatório ou opcional de cada um.

Plugin: CKEditor. O CKEditor trata-se de um editor de texto para uso diverso, destacando-se por sua versatilidade, alta customização e integração com praticamente todas as linguagens web. Com o objetivo de permitir maior personalização, realizou-se a substituição do editor WYSIWYG que trata de textos longos, como descrições, pelo editor *CKEditor 5*. Esse plugin cria um novo tipo de campo customizável, com opções pré-definidas em um arquivo de texto no diretório */config* do CMS.

Esse plugin, por sua vez, inclui outros *built-in plugins*, como mostrado na Figura 4.6. Cada elemento incorporado caracteriza determinada personalização adicionada no campo customizado, como a opção de auto formatação, negrito, itálico, lista de fontes e até inclusão de imagens e tabelas. Os campos que apresentam essa funcionalidade estão representados pelo contorno pontilhado na Figura 4.2. Ademais, essa funcionalidade apresenta interface amigável e extremamente completa, graças a inclusão do plugin[3].

[3] Exemplo do CKEditor no Strapi em https://market.strapi.io/plugins/@_sh-strapi-plugin-ckeditor

```
plugins: [
    CKEditor5.autoformat.Autoformat,
    CKEditor5.basicStyles.Bold,
    CKEditor5.basicStyles.Italic,
    CKEditor5.essentials.Essentials,
    CKEditor5.heading.Heading,
    CKEditor5.image.Image,
    CKEditor5.image.ImageCaption,
    CKEditor5.image.ImageStyle,
    CKEditor5.image.ImageToolbar,
    CKEditor5.image.ImageUpload,
    CKEditor5.indent.Indent,
    CKEditor5.link.Link,
    CKEditor5.list.List,
    CKEditor5.paragraph.Paragraph,
    CKEditor5.pasteFromOffice.PasteFromOffice,
    CKEditor5.table.Table,
    CKEditor5.table.TableToolbar,
    CKEditor5.table.TableColumnResize,
    CKEditor5.table.TableCaption,
    CKEditor5.strapiPlugins.StrapiMediaLib,
    CKEditor5.strapiPlugins.StrapiUploadAdapter,
  ],
```

Figura 4.6: *Exemplo de incorporação de plugins CKEditor*

Aplicação do contexto. Usualmente, para enviar informações de um componente pai para um componente filho, são utilizadas as *props*, no formato de herança. Resumidamente, trata-se de dados transmitidos em sentido único entre os componentes na forma de parâmetros.

```
1
2   const instituteStyle = useContext(InstituteStylesContext);
3    const barColors = [
4    instituteStyle.graphs.barChart.color1,
5    instituteStyle.graphs.barChart.color2,
6    instituteStyle.graphs.barChart.color3,
7    instituteStyle.graphs.barChart.color4,
8    instituteStyle.graphs.barChart.color5,
9    instituteStyle.graphs.barChart.color6,
10   instituteStyle.graphs.barChart.color7,
11   instituteStyle.graphs.barChart.color8,
12   ];
```

Programa 4.1: *Aplicação do Contexto*

O contexto (*Context*), por sua vez, refere-se a um método do *React* para enviar essas informações por meio de seu armazenamento na chamada *store*. Dessa forma, o componente que precisa de algum valor dentro desse conjunto de dados o busca direto da *store*, sem a necessidade de passar os valores manualmente em cada nível de componentização. Um exemplo de aplicação pode ser observado no texto de código do Programa 4.1, no qual os valores requisitados referem-se às cores dos gráficos de barras.

Essa prática é conveniente nesse cenário dado que a requisição da rota da *API* advinda do CMS encontra-se no arquivo responsável pela página inicial da plataforma. Assim, em todos os componentes nos quais algum desses valores é utilizado, seria necessário passar cada campo como *props* hereditariamente, o que tornaria o código verboso e desorganizado.

Adaptações no CSS. Como os estilos no código legado encontravam-se no formato fixo do *CSS*, foi necessário adaptar toda a lógica de atribuição de cada um a seus respectivos elementos. De forma concisa, a maior parte das modificações foram semelhantes ao exemplificado no trecho de código do Programa 4.2.

```
const Container = styled.div‘
  {...}
  background-color: ${instituteStyle.menu.backgroundColor};
  border: 1px solid;
  border-radius: ${instituteStyle.menu.borderRadius}px;
  color: ${instituteStyle.menu.fontColor};
  cursor: pointer;
  font-family: ${instituteStyle.menu.fontFamily};
  {...}
```

Programa 4.2: *Padrão de adaptação CSS*

Mudanças na geração de gráficos. A forma de requisitar os dados necessários na geração de gráficos, assim como as legendas e cores de cada um, era realizada de forma fixa e individual. Com a nova implementação, foi necessário a utilização de uma *key*, que seria enviada pela API do Strapi e preenchida no formato de campo na plataforma CMS. A variável *key* passou a ser responsável por engatilhar a requisição correta para o instituto e/ou departamento em questão. Além dessa lógica dos dados, as demais informações dos gráficos também são passadas através da mesma API.

```

const createCountByYearByDep = (productionTypeYearCountByDep) => {
  const countByYearByDep = {};
  const departments = ["AUH", "AUT", "AUP"];
  departments.forEach((dep) => {
  if (productionTypeYearCountByDep[dep]) {
    countByYearByDep[dep] = {};
    Object.values(productionTypeYearCountByDep[dep]).forEach(
      (countByYear) => {
        Object.keys(countByYear).forEach((year) => {
          const count = countByYear[year];
```

```
          countByYearByDep[dep][year] =
            (countByYearByDep[dep][year] || 0) + count;
        });
      }
    );
  }
  });
  return countByYearByDep;
};
```

Programa 4.3: *Antigo processo de geração de gráficos por departamento (FAU)*

Os trechos de código dos Programas 4.3 e 4.4 demonstram como o processo de geração de gráficos era efetuado na versão anterior. Resumidamente, a função no Programa 4.3 utiliza o *ForEach* para buscar a existência de dados no banco, referentes ao departamento em questão, para cada ano desejado. O mesmo código era replicado para cada plataforma, de forma que a única alteração consiste na declaração de *departments*, como pode ser observado no Programa 4.4. Ambos os códigos apresentam exatamente a mesma estrutura, sendo o primeiro presente na geração de gráficos da página da FAU no CPqs Abertas, enquanto o segundo descreve a geração de gráficos da FEARP.

```

const createCountByYearByDep = (productionTypeYearCountByDep) => {
  const countByYearByDep = {};
  const departments = ["RAD", "RCC", "REC"];
  departments.forEach((dep) => {
  if (productionTypeYearCountByDep[dep]) {
    {...}
  }
  });

  return countByYearByDep;
};
```

Programa 4.4: *Antigo processo de geração de gráficos por departamento (FEARP)*

Como resultado, a função representada no trecho de código do Programa 4.5 trata-se da versão adaptada para todos os institutos. Deve-se observar que enquanto anteriormente era necessário fornecer o nome de cada departamento, agora um objeto é criado previamente com o nome dos departamentos que serão utilizados, gravados como parâmetro.

```

const createCountByYearByDep = (productionTypeYearCountByDep) => {
  let countByYearByDep = {};
  let numDep = 0;
  Object.values(productionTypeYearCountByDep).forEach((dep, index) => {
  numDep++;
  index === 0 && (countByYearByDep["line1"] = {});
  index === 1 && (countByYearByDep["line2"] = {});
  index === 2 && (countByYearByDep["line3"] = {});
  index === 3 && (countByYearByDep["line4"] = {});
```

```
11      index === 4 && (countByYearByDep["line5"] = {});
12      index === 5 && (countByYearByDep["line6"] = {});
13      index === 6 && (countByYearByDep["line7"] = {});
14      });
15      let line = 1;
16      Object.values(productionTypeYearCountByDep).forEach((dep) => {
17      Object.values(dep).forEach((countByYear) => {
18       Object.keys(countByYear).forEach((year) => {
19         const count = countByYear[year];
20         line === 1 &&
21           (countByYearByDep.line1[year] =
22             (countByYearByDep.line1[year] || 0) + count);
23         line === 2 &&
24           (countByYearByDep.line2[year] =
25             (countByYearByDep.line2[year] || 0) + count);
26         line === 3 &&
27           (countByYearByDep.line3[year] =
28             (countByYearByDep.line3[year] || 0) + count);
29         line === 4 &&
30           (countByYearByDep.line4[year] =
31             (countByYearByDep.line4[year] || 0) + count);
32         line === 5 &&
33           (countByYearByDep.line5[year] =
34             (countByYearByDep.line5[year] || 0) + count);
35         line === 6 &&
36           (countByYearByDep.line6[year] =
37             (countByYearByDep.line6[year] || 0) + count);
38         line === 7 &&
39           (countByYearByDep.line7[year] =
40             (countByYearByDep.line7[year] || 0) + count);
41       });
42      });
43      line++;
44      });
```

Programa 4.5: *Nova construção da lista de valores para gerar o gráfico de linha*

No primeiro bloco de código, definido pelo *ForEach*, é computado o número de departamentos do instituto em questão e são criadas chaves, como *line1*, no objeto *countByYearByDep*. Por fim, são inicializadas como objetos vazios (*countByYearByDep["line1"]*). Em seguida, há outro bloco demarcado por três *ForEach*. Basicamente, para cada departamento, é checado cada objeto do tipo *countByYear* atribuído a ele em cada ano analisado. Assim, para cada valor de *count* encontrado, esse valor é incrementado em seu respectivo *countByYearByDep["lineX"]*, sendo X um departamento. Dessa forma, tem-se como resultado um objeto de formatação genérica que pode ser utilizado para gerar gráficos para qualquer instituto, com a limitação de conter sete departamentos. Este número limitante pode ser aumentado facilmente, mas gráficos com mais de sete linhas tornam-se visualmente confusos.

4.4 Validação

No contexto deste trabalho, a validação da nova plataforma para o CPqs Abertas foi realizada através da inclusão de uma nova unidade, o Instituto de Física (IF), com a participação ativa de um funcionário do IF que realizou os testes de aceitação. Em resumo, a facilidade para a inclusão desse novo instituto validou a nova arquitetura e confirmou a base para escalar o projeto. Enquanto a antiga arquitetura demandava um esforço de replicação e adaptação de um código existente, o que demandava bastante tempo, a nova estrutura permitiu a inclusão e o lançamento de um novo site de um instituto no CPqs Abertas em poucas horas.

Figura 4.7: *Física*

O primeiro passo para a inserção do novo instituto foi o povoamento do banco de dados com as informações dos currículos dos professores do Instituto de Física, obtidos via a Plataforma Lattes do CNPq. Após isso, apenas foi necessário criar os módulos de estilos no Strapi e gerar a implantação do código para que pudesse ser enviado para o serviço S3 da Amazon, gerando o novo site, conforme ilustrado na Figura 4.7.

Os testes com a inclusão do Instituto de Física guiaram a identificação dos problemas de interface de usuário. Essas informações foram importantes para direcionar os ajustes finais antes da implementação completa. Com base nas percepções e nas experiências dos testadores (funcionário do IF e equipe de designers), foram realizadas modificações que não apenas corrigiram as questões identificadas para o IF, mas também forneceram uma melhor direção para o futuro do projeto e para os próximos passos a serem seguidos. Este ciclo de ajustes ajudou a assegurar que a implementação completa do sistema atendesse às expectativas e necessidades de um primeiro grupo de usuários.

Entre as modificações estavam a adição de alguns campos antes não componentizados

pela equipe de desenvolvimento, mas observados como importantes pelos designers. Contudo, as principais modificações apontadas se referiam à interface inicial do projeto, que apresentava diversos problemas, citados a seguir:

- Fluxo de navegação complexo: muitas das informações eram dificilmente encontradas pelo usuário, como a produção artística por departamento.
- Questões de usabilidade em relação às informações quantitativas: muitos dos valores numéricos não possuíam unidade de medida, ou não indicavam a qual departamento essas medidas se referiam.
- Questões de usabilidade em relação às informações qualitativas: problemas com palavras desconexas na nuvem de palavras.
- Sobrecarga visual: a presença de muito texto em diversas páginas pode levar a confusão do usuário ou desincentivar que ele leia a informação escrita.
- Disposição confusa e não intuitiva de elementos: especialmente em relação ao menu, responsável por todo fluxo da página, que estava disposto de forma ineficiente.
- Ausência de ligação com as demais plataformas: não havia um link que direcionasse o usuário para a página "principal" do projeto, o Hub do CPqs Abertas, o qual possui um link para todas as plataformas já criadas, assim como um formulário de interesse.

Por conta do prazo para conclusão deste trabalho de conclusão de curso, a equipe (autores deste trabalho) não implementou nenhuma das melhorias citadas. Contudo, com base nas sugestões dos designers, foram elaboradas tarefas específicas que irão direcionar o próximo estágio do projeto. Essas tarefas foram documentadas de maneira detalhada, visando possibilitar um rápido entendimento pelas próximas pessoas desenvolvedoras que irão evoluir a nova plataforma do CPqs Abertas. Além disso, a documentação foi estruturada para permitir o tratamento ágil das urgências que possam surgir, garantindo a capacidade de resposta diante dos desafios ou ajustes necessários. Esse conjunto de tarefas não apenas delineia o caminho a seguir, mas também oferece uma abordagem flexível para lidar com as demandas, mantendo um fluxo eficiente e adaptável no prosseguimento do projeto.

4.5 Perspectivas futuras

Com a validação e implantação da nova plataforma do CPqs Abertas, surgiram direcionamentos para o futuro do projeto, a partir dos testes com usuários e avaliação dos designers (bolsistas da FAU vinculados ao projeto). As mudanças sugeridas pelos designers auxiliaram na definição de prioridades e construção de um roteiro para os próximos passos. Com base nos testes dos usuários, foram criadas tarefas (issues no repositório do projeto no Gitlab) específicas. A melhoria contínua do projeto não apenas visa corrigir problemas identificados, mas também estabelecer uma base sólida para a evolução do projeto, em especial porque também se trata de um software livre.

Com a nova plataforma, espera-se inserir todos os institutos da USP no CPqs Abertas no médio prazo, ampliando o seu uso e reconhecimento ao longo do tempo. Com a nova arquitetura, o processo de criação de um novo site para outro instituto ficou facilitado;

no entanto, ainda há alguns pontos a serem melhorados para otimizar e aprimorar a infraestrutura do projeto. Os próximos passos serão muito importantes para a ampliação do CPqs Abertas de forma escalável e manutenível, exigindo comprometimento dos próximos desenvolvedores do projeto. Os próximos passos sugeridos estão descritos seguir, em uma ordem de prioridade para a evolução do projeto:

1. Como o objetivo do projeto é inserir um número alto de institutos, o processo de inserir um novo site ou atualizar os existentes na AWS pela S3 deve ser automatizado, utilizando técnicas de implantação e integração contínua.
2. Separação do container do MongoDB, retirando esse componente do back-end e introduzindo um novo tipo de tecnologia que armazena esse tipo de banco de dados.
3. Automatizar o processo de povoamento do banco de dados quando um novo instituto for inserido.
4. Introduzir técnicas de implantação e integração contínua para o novo back-end.
5. Componentização do Strapi, ou seja, organizar os diferentes tipos de módulos de estilos, para que os designers que utilizarão a plataforma entendam melhor o seu funcionamento.
6. Refatoração do código-fonte do front-end, com o objetivo de facilitar a implementação de novos recursos e alterações, como o tipo de menu dos sites, sugeridas pelos designers do projeto.

Essa lista ordenada de tarefas foi pensada para cobrir os principais pontos que o projeto precisa evoluir prioritariamente, os quais, mesmo com a nova arquitetura, não foram implementados pela equipe por questão de escopo dentro do prazo estabelecido para a finalização deste trabalho de conclusão de curso.

Capítulo 5

Conclusão

Este estudo buscou analisar a questão da escalabilidade sob a perspectiva da estratégia "White Label" + *Headless* CMS, tendo como objeto de estudo o projeto CPQs Abertas. Ao longo do período de desenvolvimento, foi possível comprovar os impactos da migração de arquitetura em relação à redução de custos. Esses resultados oferecem visões mais concretas e um futuro mais palpável ao projeto.

Levando em conta a relevância de produções científicas de docentes e discentes na Universidade e, considerando seu papel como órgão público, a transparência e divulgação dos trabalhos para o desenvolvimento do país é necessária. Por isso, no contexto do CPqs Abertas, sendo este o que busca justamente facilitar a divulgação citada, os objetivos deste trabalho discutidos no Capítulo 1 se mostram justificáveis e relevantes já que a Universidade de São Paulo além de contar com o seu campus na capital, possui unidades espalhadas por todo o estado de São Paulo.

Ao considerar as limitações deste trabalho, no contexto do desenvolvimento de sistemas a presença de dificuldades no decorrer do período é imprescindível. Contudo, o caráter adaptativo do projeto levou a resultados preferíveis, o que destaca a importância do modelo ágil no desenvolvimento. Entre as mudanças de maior inovação postas como essenciais na projeção futura de desenvolvimento, estão a nova disposição dos elementos de forma a respeitar os conceitos de *User Interface Design* (UI) e garantir a melhor experiência no que diz respeito à experiência visual da plataforma.

A relevância deste estudo vai além do aprimoramento do projeto, garantindo uma contribuição para a estratégia relativamente recente de aplicação "White Label" + *Headless* CMS em Software. Dessa forma, foi possível averiguar que a implementação de uma arquitetura pautada nesse conceito e sua compreensão aprofundada garantem inúmeros benefícios no que diz respeito, sobretudo, à escalabilidade e facilidade. Portanto, neste trabalho, validou-se a estratégia citada, principalmente com a inserção do Instituto de Física da USP, o que permitiu uma perspectiva positiva para o futuro do CPQs Abertas, pois, com o que foi desenvolvido, seu crescimento tornou-se mais factível, permitindo de maneira menos custosa e ágil, a integração de novas unidades.

Em última análise, o trabalho realizado trouxe consideráveis benefícios para o projeto

de estudo, tendo em vista os valores e objetivos atrelados a ele, tais como boas práticas de programação no contexto de desenvolvimento de software já que, com a estratégia abordada foi possível solucionar o problema de códigos duplicados para cada unidade participante colaborando assim com futuras manutenções do software. É incontestável que as mudanças realizadas forneceram a base para obtenção dos objetivos finais instalados, com destaque para a escalabilidade do sistema. Ademais, a transição trouxe também vantagens em relação à evolução como um projeto de software livre propriamente dito. Espera-se que este trabalho contribua para o desenvolvimento contínuo do CPqs Abertas e, por conseguinte, da divulgação da produção docente não somente no ambiente acadêmico, mas na sociedade em geral.