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

CPqs Abertas: Desenvolvimento de um sistema modularizado e adaptável para exposição das produções acadêmicas da USP

João Gabriel Loureiro de Lima Lembo,
Leonardo Alves Pereira, Victor Pereira Lima

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Alfredo Goldman Vel Lejbman
Cossupervisor: Bacharel João Francisco Lino Daniel

Durante o desenvolvimento deste trabalho os autores João Gabriel Lembo e Victor Lima receberam auxílio financeiro da Pró-Reitoria de Pesquisa da Universidade de São Paulo

São Paulo
24 de Dezembro de 2021

Agradecimentos

Primeiramente gostaríamos de agradecer a todos os nossos amigos e familiares, que estiveram do nosso lado durante todo o percurso da graduação e da execução deste trabalho.

Agradecemos também ao professor Alfredo Goldman Vel Lejbman e ao mestrando João Francisco Lino Daniel, que foram de fundamental importância, nos orientando e auxiliando durante toda a elaboração deste trabalho.

Agradecemos ao professor Artur Rozestraten, ao doutorando Luiz Felipe Abbud e ao graduando Gustavo Alves Machado, que compuseram a equipe do projeto CPqs Abertas junto conosco durante o ano.

A todos os professores do Bacharelado em Ciência da Computação da USP, que nos proporcionaram os conhecimentos e ferramentas necessários para o desenvolvimento deste trabalho.

Agradecemos aos membros da FAU e da FEA-RP, os clientes deste projeto, que se mostraram sempre à disposição para nossos questionamentos.

A todos aqueles que tiveram a iniciativa do projeto CPqs Abertas, aos que confiaram neste para a divulgação de seus trabalhos, às equipes anteriores do projeto e às equipes que continuarão este trabalho no futuro.

Agradecemos, por fim, a todos aqueles que de alguma forma nos auxiliaram durante toda a trajetória, tanto em relação a este trabalho quanto a aspectos externos, o que permitiu que tivéssemos oportunidade, calma e seriedade para o desenvolvimento deste trabalho.

Resumo

João Gabriel Loureiro de Lima Lembo, Leonardo Alves Pereira, Victor Pereira Lima. **CPqs Abertas: Desenvolvimento de um sistema modularizado e adaptável para exposição das produções acadêmicas da USP**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Este trabalho tem como objetivo apresentar o projeto CPqs Abertas e exibir todas as atividades relacionadas ao desenvolvimento realizadas durante o ano. O trabalho consistiu na elaboração de um sistema modularizado e adaptável, que pudesse ser expandido de forma simples e rápida para outros institutos da USP, visando divulgar suas produções acadêmicas. O desenvolvimento teve principalmente duas fases, sendo a primeira a de manutenção e criação de novas funcionalidades para o FAU Aberta, primeiro sistema web do projeto, e a segunda a da modularização do sistema para permitir sua expansão. Esta modularização consiste na separação do sistema em diversos componentes que podem ser usados para construir novos sistemas web rapidamente, a partir das necessidades e solicitações de cada instituto. Permearam o trabalho as boas práticas de arquitetura de software e de métodos ágeis, com comunicação constante entre os envolvidos no projeto, e refatorações e melhorias frequentes no código. Durante o trabalho, foi possível modularizar o sistema de forma eficaz, algo evidenciado pela velocidade com que se pôde expandi-lo para a FEA-RP, segunda unidade participante do projeto. No momento da conclusão deste trabalho, além do FAU Aberta e do FEA-RP Aberta, já finalizados, há também o IME Aberto, em processo de desenvolvimento. Há também diversos institutos que já mostraram interesse e desejam integrar o projeto CPqs Abertas. Espera-se que no futuro novas equipes possam continuar o desenvolvimento que foi realizado, aproveitando-se do sistema modularizado para criar essas novas unidades de forma rápida e eficaz.

Palavras-chave: Modularização. Producões Acadêmicas. Refatoração. Métodos Ágeis.

Abstract

João Gabriel Loureiro de Lima Lembo, Leonardo Alves Pereira, Victor Pereira Lima. **CPqs Abertas: Development of a modularized and adaptable system for the exhibition of academic productions from USP**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

The objective of this monograph is to present the CPqs Abertas project and exhibit all the system development-related activities carried out during this year. The task consisted in the development of a modularized and adaptable system which could be expanded in a simple and rapid manner to other institutes at USP, aiming to disseminate their academic productions. The development was mainly composed of two phases, the first being maintenance and the creation of new functionalities to FAU Aberta, the project's first web system, and the second being the modularization of the system to allow its expansion. This modularization consisted in the separation of the system into multiple components that can be used to build new web systems rapidly, based on the needs and requests of each institute. The execution of the project was driven by the best practices of software architecture and agile methods, with constant communication between those involved, and frequent refactoring and improvement of the code. During the development, it was possible to modularize the system effectively, something evidenced by the quick expansion to FEA-RP, the second unit to participate in the project. At the time of conclusion of this monograph, besides FAU Aberta and FEA-RP Aberta, already finished, there is also IME Aberto, in development process. There are also several institutes that have already shown interest in being part of the project. It is expected that future teams can continue the development, taking advantage of the modularized system to expand to other institutes quickly and efficiently.

Keywords: Modularization. Academic Productions. Refactoring. Agile Methods.

Lista de Abreviaturas

CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
CPq	Comissão de Pesquisa
FAU	Faculdade de Arquitetura e Urbanismo
FDRP	Faculdade de Direito de Ribeirão Preto
FEA-RP	Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto
HTTP	Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*)
IAU	Instituto de Arquitetura e Urbanismo
IF	Instituto de Física
IME	Instituto de Matemática e Estatística
ORM	Mapeamento Objeto-Relacional (*Object-Relational Mapping*)
PRP	Pró-Reitoria de Pesquisa
SQL	Linguagem de Consulta Estruturada (*Structured Query Language*)
STI	Superintendência de Tecnologia da Informação
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
USP	Universidade de São Paulo

Lista de Figuras

Lista de Tabelas

Lista de Programas

Sumário

Capítulo 1

Introdução

1.1 Motivação

A produção acadêmica das universidades permanece constante e ativa, e é de fundamental importância para a sociedade. Seja qual for a área do conhecimento, ampliar o entendimento dela frequentemente é o que permite que a sociedade melhore, cresça e evolua. Entretanto, não basta apenas que as produções continuem sendo feitas, mas sim que elas sejam divulgadas e alcancem boa parte da população.

Infelizmente, entretanto, essa divulgação ainda não é feita de forma ampla e eficiente, em muitos casos. Muito pouco do que se produz dentro da universidade é exposto para o lado de fora de suas fronteiras, muitas vezes sendo pouco divulgado inclusive entre institutos de uma mesma universidade, ou até para os outros membros do mesmo instituto.

Dessa forma, muitas pessoas não conhecem o que é construído dentro da universidade. É o que mostra, por exemplo, uma pesquisa realizada pelo Instituto Nacional de Comunicação Pública da Ciência e Tecnologia com jovens entre 15 e 24 anos, a qual evidenciou que apenas 12% desses jovens sabia citar ao menos uma instituição que realizasse pesquisa científica (Escobar, 2019).

Na Faculdade de Arquitetura e Urbanismo da USP (FAU) o cenário não era diferente, e alguns de seus professores buscaram meios de não apenas divulgar mais suas produções, mas também fazê-lo de forma agradável e amigável ao usuário. Assim surgiu o projeto FAU Aberta, que pretendia criar uma página web que pudesse reunir na Internet os dados sobre a produção intelectual da FAU e divulgá-los de forma didática, acessível e qualitativa.

Desta maneira, o FAU Aberta apresenta-se como uma plataforma que visa principalmente a exposição da produção intelectual da FAU tanto para dentro quanto para fora da universidade. Assim, divulga-se cada vez mais o conhecimento e também se mostra o potencial acadêmico da universidade pública de forma quantitativa e qualitativa, através de gráficos, mapas e outras funcionalidades. Sendo assim, além de expor toda essa produção, também o faz de forma simples, muito bem legível e moderna, tornando o acesso mais fácil e mais convidativo, possibilitando uma divulgação ainda melhor de todos os dados.

1.2 Contexto

Com este projeto em mente, a FAU pôde contar com a contribuição do professor do Instituto de Matemática e Estatística (IME) Alfredo Goldman (orientador deste trabalho), que sugeriu a utilização do scriptLattes, uma ferramenta para extração de dados dos currículos da base Lattes-CNPq, desenvolvida pelo professor Jesús Pascual Mena-Chalco, da UFABC, em 2015.

Assim, no segundo semestre de 2019, sob a coordenação da professora Beatriz Bueno, presidente da Comissão de Pesquisa da FAU (CPq-FAU) na época, o FAU Aberta passou a ser de fato desenvolvido, em primeira versão, na disciplina de graduação Laboratório de Programação Extrema (MAC0342), sendo continuado no segundo semestre de 2020, nesta mesma disciplina.

Então, no início do ano de 2021, os professores que coordenaram o projeto, Artur Rozestraten, presidente da CPq-FAU; Beatriz Bueno, vice-presidente da CPq-FAU; e Alfredo Goldman, presidente da CPq-IME, decidiram ampliar seu escopo, buscando oferecer, a outros institutos, sistemas que pudessem divulgar suas produções científicas com qualidade e acessibilidade. Deu-se assim o surgimento do projeto CPqs Abertas, com o objetivo de ampliar o que foi feito na FAU, desenvolvendo ferramentas similares para os outros institutos da USP.

1.3 Objetivos

Desta forma, tem-se que os objetivos deste trabalho foram dar continuidade ao projeto FAU Aberta, tanto implementando novas funcionalidades quanto fazendo as manutenções necessárias; desenvolver um sistema altamente modularizado, que permite uma extensão rápida do sistema para outros institutos; estudar a arquitetura do FAU Aberta e as boas práticas de arquitetura de software; estender o sistema para a Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto (FEA-RP) e para o IME.

1.4 Metodologia

A equipe do projeto CPqs Abertas, no ano de 2021, foi composta pelos coordenadores do projeto, Alfredo Goldman e Artur Rozestraten, pelo estudante de design[1] Gustavo Machado, sob supervisão do doutorando Luís Felipe Abbud e pelos três autores deste trabalho, que compuseram a equipe de desenvolvimento, sob supervisão do mestrando João Daniel. O projeto foi elaborado seguindo as práticas da metodologia ágil, com comunicação constante entre as partes envolvidas, desenvolvimento em pequenas etapas, frequente melhoria e aprimoramento do que está sendo desenvolvido, entre outros. O repositório utilizado durante este projeto pode ser encontrado em https://gitlab.com/availablenick/cpqs-abertas.

[1] No decorrer deste trabalho, opta-se por não escrever design e suas variações em itálico, visto que a palavra já está incorporada na Língua Portuguesa.

Inicialmente, foram estudadas de maneira mais aprofundada as práticas da arquitetura de software, através do livro *Clean Architecture: A Craftsman's Guide to Software Structure and Design* (Martin, 2017), visando entender melhor seus conceitos para poder aplicá-los de forma eficiente durante o projeto. Estes estudos permearam todo o período de desenvolvimento do trabalho, buscando técnicas e ferramentas que pudessem solucionar os problemas de arquitetura encontrados durante a elaboração do projeto.

Em seguida, lidando com o projeto de forma mais direta e prática, foi examinado o código já existente do FAU Aberta, buscando entendê-lo para que pudessem ser feitas modificações a pedido da FAU. Além disso, foram analisados os pontos que poderiam ser aproveitados para desenvolvimento do trabalho e da modularização, além daqueles que precisariam ser modificados, melhorados ou até refeitos.

Assim, iniciou-se o desenvolvimento propriamente dito, em que primeiramente foram feitas algumas modificações na arquitetura do software, seguidas pela realização de algumas solicitações da FAU. Em seguida realizou-se a modularização de fato, tanto no *front-end*[2] quanto no *back-end*[3], para permitir a criação do sistema para outros institutos. Foi depois dessa etapa que se pôde criar o sistema FEA-RP Aberta, utilizando os componentes comuns e fazendo as modificações necessárias, e também preparar o sistema para a criação do IME Aberto.

Esse desenvolvimento iniciou-se com *Mob Programming*[4], realizado três vezes por semana. Sendo assim, foi realizado um revezamento entre os três membros da equipe de desenvolvimento, em que a cada encontro um era responsável por compartilhar sua tela e digitar o código. Enquanto isso, os outros membros o auxiliavam buscando soluções para os problemas encontrados, discutindo estratégias e minimizando a ocorrência de erros devido ao maior número de pessoas observando o mesmo código.

Esta técnica foi utilizada pois garantia que os três estivessem sempre em contato com o código, de forma que todos pudessem entender o que estava sendo feito e de que forma. Isso era ainda mais importante no início do projeto, pois permitia que todos estudassem juntos o código do FAU Aberta e se familiarizassem com ele. Além disso, constantemente era necessário tomar decisões de projeto importantes, e ter todos os membros do grupo envolvidos era essencial.

No último trimestre de desenvolvimento, as tarefas começaram a ser divididas entre os três membros da equipe, com uma reunião semanal onde o trabalho de cada um era revisado pelos demais e novas tarefas eram distribuídas. Isso foi feito para acelerar o desenvolvimento, principalmente por conta da demanda de finalizar o sistema de um novo instituto, visto que este seria crucial para verificar se as modularizações foram de fato eficientes. Ademais, todos já estavam muito familiarizados com o código e os próximos passos eram sempre decididos em conjunto, e, portanto, realizar todos os encontros de *Mob Programming* não era mais necessário.

Já na questão do planejamento, reuniões foram feitas quinzenalmente, entre toda a

[2] Ver **Apêndice A**.

[3] Ver **Apêndice A**.

[4] Ver **Apêndice A**.

equipe do CPqs Abertas, para discutir o andamento, decidir novos passos e organizar o projeto. Além disso, foram realizadas reuniões com membros da FAU durante o desenvolvimento do FAU Aberta, e com membros da FEA-RP para discussão do FEA-RP Aberta.

1.5 Cronograma

Para melhor compreensão da execução das atividades supracitadas, apresenta-se um cronograma explicitando os meses em que foram realizadas:

Atividades	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
Estudo de código e arquitetura	×	×	×	×	×	×	×	×	
Desenvolvimento		×	×	×	×	×	×	×	×
Monografia							×	×	×

Tabela 1.1: *Cronograma do trabalho.*

É importante ressaltar que o estudo e o desenvolvimento não tiveram cronograma detalhado de atividades por conta dos métodos ágeis empregados durante a execução do projeto. Desta forma, as tarefas específicas a serem feitas eram decididas nas reuniões quinzenais e nas sessões de *Mob Programming* realizadas pela equipe.

1.6 Estrutura dos capítulos

O capítulo 2, **Conhecendo o Código**, apresenta detalhes técnicos do sistema no primeiro contato da equipe. O capítulo 3, **Atividades de Desenvolvimento**, detalha as atividades realizadas durante o ano. O capítulo 4, **Interações da Equipe**, discorre sobre as interações entre a equipe de desenvolvimento e outros grupos, como a equipe de design e os clientes. A visão pessoal de cada autor deste trabalho em relação a essas interações é apresentada no capítulo 5, **Experiências Pessoais**. Por fim, no capítulo 6, **Próximos Passos e Conclusão**, são expostos alguns próximos passos possíveis em relação ao projeto e uma conclusão sobre os objetivos deste trabalho e o estado atual do projeto.

Capítulo 2

Conhecendo o código

Este capítulo descreve o estado do código no primeiro contato da equipe, analisado em algumas reuniões ao final do mês de maio, explicitando a estrutura do projeto e as ferramentas utilizadas.

2.1 Estrutura Geral

O sistema estava, resumidamente, dividido em três partes: o banco de dados, o *back-end* e o *front-end*. Além disso, o projeto possuía uma infraestrutura que permitia que cada uma de suas partes fosse executada em contêineres, utilizando virtualização em nível de sistema operacional, facilitando tanto o desenvolvimento quanto o *deploy*[1].

A figura a seguir exibe, de forma geral, a estrutura do sistema:

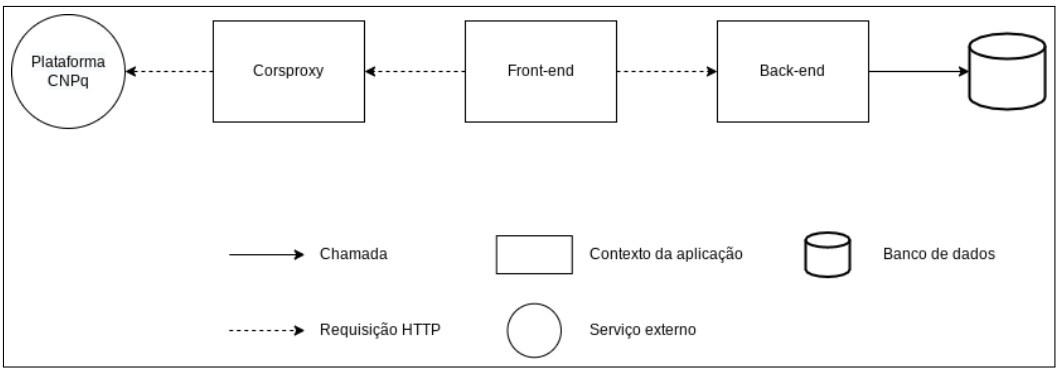

Figura 2.1: *Estrutura geral do FAU Aberta.*

Todo o código do sistema a que esse capítulo se refere está hospedado no GitLab, e pode ser acessado através do link https://gitlab.com/fau-aberta/fau-aberta.

A seguir, são apresentadas tanto a infraestrutura quanto as três partes do sistema de forma mais detalhada, esclarecendo a função de cada uma delas e as tecnologias utilizadas.

[1] Ver **Apêndice A**.

2.2 Infraestrutura

Como mencionado anteriormente, cada uma das partes do sistema estava separada em um pacote, chamado de contêiner. Essa tecnologia de conteinerização garante que cada um desses pacotes seja autocontido, de forma que todo o código, arquivos de configuração e bibliotecas de um software estejam em seu próprio contêiner. Além disso, eles podem se comunicar, garantindo que todos os componentes do sistema conversem entre si.

Um desses benefícios, que inclusive é um dos mais importantes, é o de garantir a reprodutibilidade do ambiente. Isto é, ao executar o sistema em uma nova máquina, não é necessário instalar bibliotecas do projeto e criar muitos arquivos de configuração, pois esses já se encontram nos contêineres. Desta forma, fica assegurado que todas as máquinas em que eles são executados possuam o mesmo ambiente para a execução do sistema.

Sendo assim, o desenvolvimento é facilitado, já que os programadores não precisam se preocupar em garantir que seus ambientes estejam configurados da mesma maneira. O *deploy* é igualmente favorecido, já que também não existe esse trabalho de garantir o ambiente igual na máquina em que o sistema estará hospedado. Isso é ainda mais útil quando o sistema vai ser executado na nuvem ou em alguma máquina de difícil acesso, pois reduz muito o tempo gasto com configurações que devem ser feitas antes de se executar o sistema.

A ferramenta utilizada pelo FAU Aberta para a conteinerização foi o Docker, que permite a execução dos contêineres em Linux, Windows ou macOS, tanto em redes locais quanto na nuvem. A criação deles é feita através de uma imagem, que consiste de um modelo com instruções especificando uma imagem base e o modo como a construção do contêiner deve ser feita. Essas imagens são definidas através de um Dockerfile, um arquivo de texto que contém todos os comandos para configurar e montar uma imagem.

Para facilitar a construção da aplicação, foi utilizado o Docker Compose, uma ferramenta para definir e executar aplicações Docker com múltiplos contêineres. Esta ferramenta permite a automatização de todo o processo de inicialização e serve também como uma forma de documentação da estrutura do sistema, através do arquivo Compose. Esse é um arquivo YAML utilizado pelo Docker Compose para configurar os serviços da aplicação. Nele, é possível definir dependências entre os serviços, suas comunicações, entre outros. Depois disso, com apenas um comando, é possível inicializar todos esses serviços a partir dessa configuração.

Desta forma, o projeto do FAU Aberta apresentava quatro Dockerfiles, para o *front-end*, o *back-end*, o banco de dados e um pequeno servidor *proxy*[2], determinando a imagem base e os comandos que deveriam ser executados para que essas partes do sistema fossem implantadas; e o Compose, que estabelecia a relação entre os contêineres criados a partir dos Dockerfiles e permitia que o sistema fosse executado.

[2] Ver **Apêndice A**.

2.3 Banco de dados

O banco de dados é a parte do sistema responsável por armazenar todos os dados necessários para a sua execução. No caso do FAU Aberta, esses dados são referentes aos professores da FAU e suas produções acadêmicas, obtidos na plataforma Lattes do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). Nesta etapa do projeto, esses dados estavam sendo coletados manualmente, através de downloads na página de cada professor na plataforma, e inseridos no banco de dados através de um *parser*[3] dos currículos criado pela equipe do FAU Aberta.

O sistema gerenciador de banco de dados utilizado foi o PostgreSQL, um sistema de código aberto que administra bancos de dados do tipo relacional, a partir da linguagem SQL. Esse tipo de banco armazena todos os seus dados em tabelas, sendo cada uma delas uma entidade, na qual as colunas representam os atributos e as linhas representam os objetos dessa entidade. Além disso, existem tabelas que representam a relação entre diferentes entidades, fazendo com que certos registros de uma tabela se associem a registros de outra.

Desta forma, o banco de dados do sistema era composto de múltiplas tabelas, para professores, cada tipo de produção, e as relações entre estes, permitindo que todas as informações relevantes pudessem ser exibidas no sistema.

2.4 *Back-end*

O *back-end* é a parte do sistema responsável pela comunicação com o banco de dados, de forma a buscar informações específicas de acordo com diferentes ações do sistema. Sendo assim, é ele o responsável por fazer diferentes consultas no banco de dados, tratando-os de diferentes formas, para fornecer dados específicos em um formato pronto para ser consumido por outras partes do sistema. No caso do FAU Aberta, quem requisitava os dados era o *front-end*.

Esse tratamento de dados é feito a partir das regras de negócio, de forma que estas se concentram fortemente no *back-end*. Portanto, mesmo que outras partes do sistema tenham alguns tipos de filtro ou tratamento, as regras essenciais são colocadas no *back-end*.

A ferramenta utilizada na sua criação foi o Django, um *framework*[4] para desenvolvimento web feito para a linguagem de programação Python. Esse *framework* já oferece uma estrutura web pronta, acelerando muito o desenvolvimento e permitindo que os programadores se concentrem nos elementos específicos de seus sistemas.

Dentro dessa estrutura web, o Django fornece uma forma prática e fácil de criar uma API web para realizar a transmissão de dados necessária para o sistema. A API é uma interface que permite a comunicação entre diferentes programas, e, no caso da web, possui um formato de requisição-resposta. Isto é, um programa faz uma requisição a outro através da API web, e este então retorna uma resposta ao requisitante.

---

[3] Ver **Apêndice A**.

[4] Ver **Apêndice A**.

Por conter uma estrutura que fornece essa criação de maneira simples, o Django possibilita que se crie uma interface que permita que uma outra parte do sistema faça solicitações específicas. Estas são feitas a partir de rotas, que são acessíveis por URLs (*Uniform Resource Locator*), que retornam diferentes dados de acordo com as requisições feitas.

Além disso, o Django possui ferramentas para utilização da técnica ORM (*Object-Relational Mapping*), Mapeamento Objeto-Relacional, que possibilita um mapeamento entre classes do Python e relações do banco de dados. A partir desse mapeamento, a manipulação dos dados torna-se muito menos complexa e as consultas e atualizações no banco de dados são facilitadas.

Em relação à estrutura, o Django é, em termos gerais, composto de um conjunto de *Models*[5], que são os modelos que representam cada entidade, e de múltiplos apps definidos pelos desenvolvedores.

Os modelos são classes do Python que definem quais são as entidades e quais são seus atributos. São essas classes que permitem que o Django utilize o ORM, pois ele automaticamente identifica o mapeamento entre as classes e as entidades no banco de dados.

Já os apps são conjuntos de arquivos que determinam certas ações do sistema. No caso do FAU Aberta, cada app era composto principalmente por três arquivos: *urls.py*, *views.py* e *tests.py*. O primeiro é responsável por determinar quais são as rotas disponíveis naquele app; o segundo por determinar quais ações serão tomadas quando uma rota for acessada; e o terceiro por conter todos os testes referentes às rotas e ações determinadas nos outros arquivos.

Por fim, o Django possui um app padrão, o qual contém todas as configurações do sistema, como endereço do banco de dados e quais os apps o compõem, e integra as rotas de todos os apps sobre uma URL comum.

A estrutura do *back-end* é apresentada na figura a seguir:

[5] Ver **Apêndice A**.

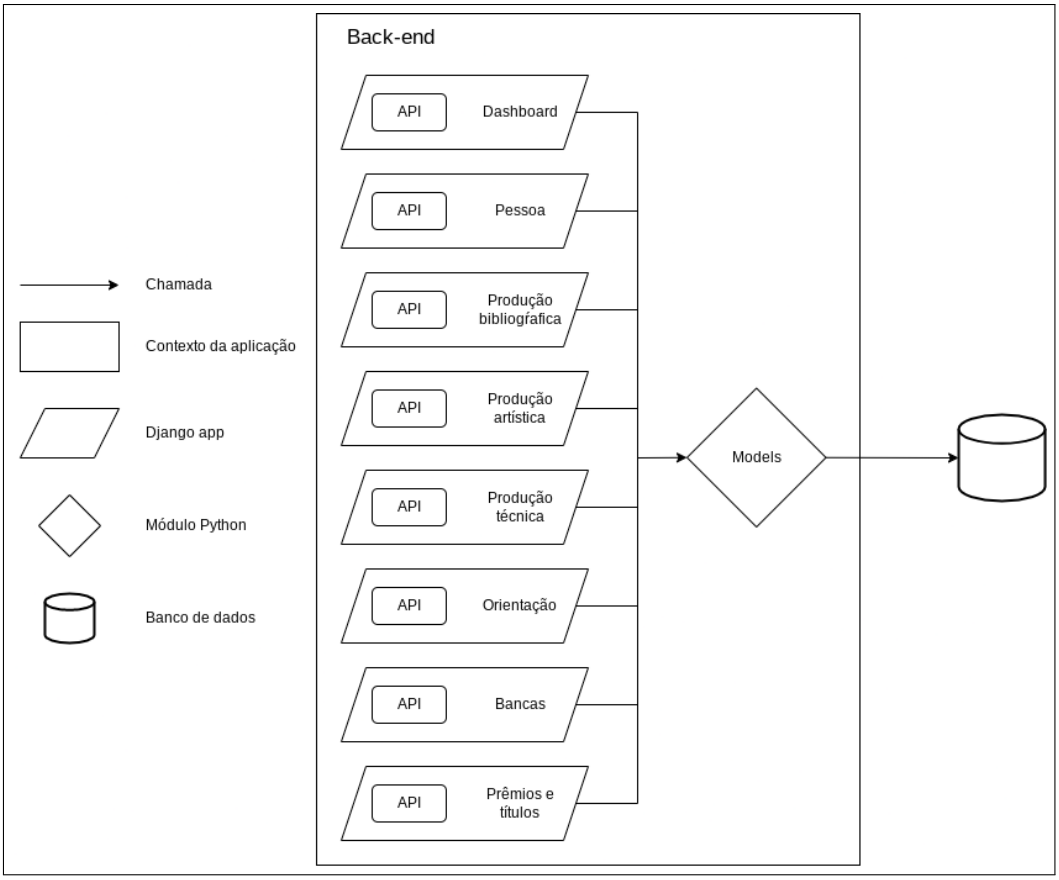

Figura 2.2: *Estrutura do back-end do FAU Aberta.*

2.5 *Front-end*

O *front-end* é a parte do sistema que está em contato direto com o usuário. Ou seja, é a parte em que os dados do sistema são exibidos de forma gráfica e interativa, permitindo ao usuário navegar pelas diferentes seções, buscar diversas informações e de fato ter acesso a tudo que a FAU pretendia que fosse divulgado.

A linguagem de programação utilizada foi o JavaScript, muito comum no desenvolvimento web, permitindo a interação do usuário através do navegador web. Esta linguagem foi usada através do Node.js, um ambiente para desenvolvimento *server-side*[6] para programação com JavaScript, que permite sua execução mesmo fora de um navegador web. Assim, é capaz de criar páginas dinâmicas antes de enviá-las ao browser, produzindo páginas customizadas de acordo com as ações do usuário.

Além disso, sobre essa linguagem e esse ambiente estava o React, uma biblioteca para desenvolvimento web baseada em componentes, que facilita não só o gerenciamento de estados dentro da aplicação, mas também a criação de páginas dinâmicas e interativas.

A aplicação criada pelo React é uma aplicação de página única (SPA, do inglês *Single Page Application*), o que significa que os conteúdos da página acessada são modifica-

[6] Ver **Apêndice A**.

dos de forma dinâmica conforme as ações do usuário, ao invés de páginas novas serem geradas e carregadas a cada interação. Isso fornece uma experiência muito melhor ao usuário, diminuindo o tempo de carregamento e tornando as transições entre páginas mais rápidas.

O React é majoritariamente constituído de componentes. Nesse contexto, um componente é uma função ou classe (pode-se usar as duas implementações) que define os elementos a serem exibidos em uma página e quaisquer processamentos necessários, como a requisição de dados para o *back-end*. Esses componentes podem receber parâmetros para sua criação, e tanto estes quanto os atributos do próprio componente podem ser modificados a qualquer hora, atualizando instantaneamente o que é exibido na tela. É justamente esse comportamento que permite o dinamismo das páginas.

Os componentes do React podem representar tanto uma página do sistema, como apenas um pedaço dela. Sendo assim, eles podem ser aninhados, de forma que um componente que representa uma página seja constituído de vários deles, com estes também podendo ser constituídos de vários outros. No sistema, existe também um componente App, que representa o sistema como um todo e é o responsável por determinar quais rotas web exibem quais componentes.

Por fim, é interessante ressaltar que também existiam trechos de código nas linguagens HTML e CSS, que permitiam, respectivamente, a estruturação e a estilização das páginas web criadas pelo sistema.

Na figura a seguir exibe-se a estrutura do *front-end*:

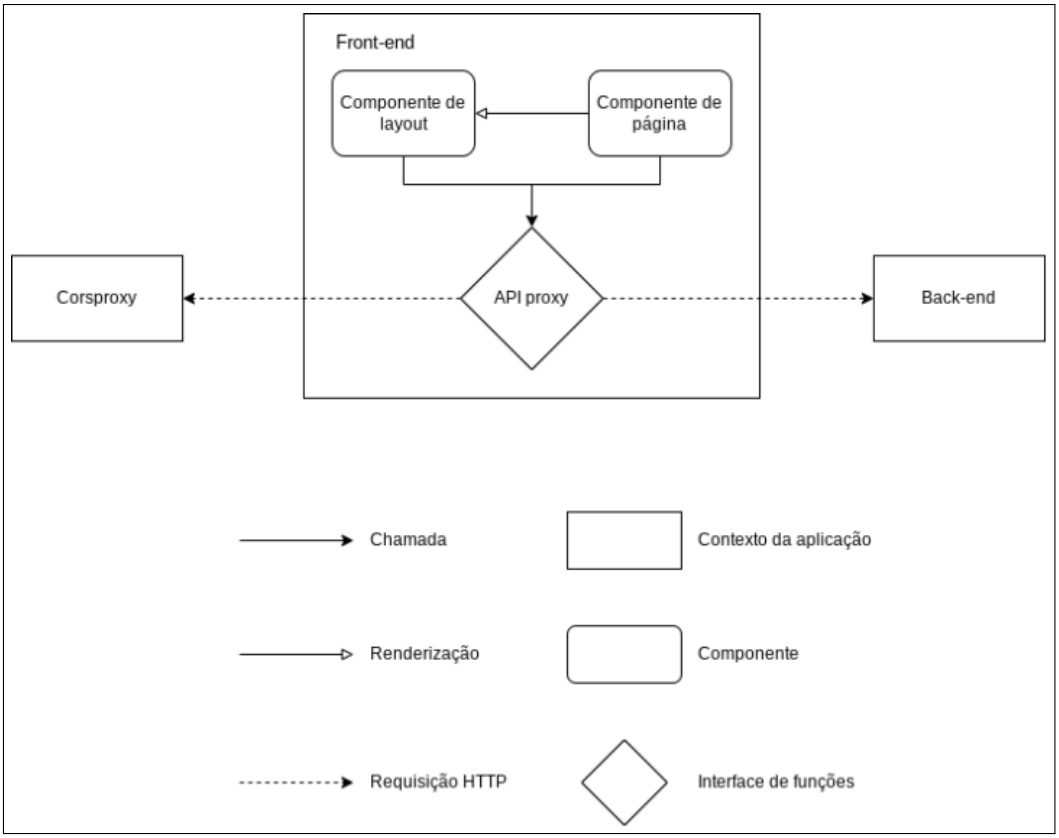

Figura 2.3: *Estrutura do front-end do FAU Aberta.*

2.6 Corsproxy

O corsproxy era um pequeno servidor *proxy* utilizado para fazer conexões com o site do CNPq, para obter a imagem de perfil dos professores do sistema. Era composto de um Dockerfile simples, que executava o servidor, através de uma biblioteca do Node.js.

2.7 Interação dos componentes

Evidenciadas todas as partes do sistema, é possível entender melhor como elas interagiam entre si. Como mencionado, o banco de dados era carregado com informações dos professores e suas produções acadêmicas a partir de um *parser* que lia os currículos em formato XML. A partir disso, com o sistema em execução, o usuário podia acessar o *front-end*, que era composto de diversas páginas web (em sua maioria, dinâmicas) que exibiam diversos desses dados das mais diferentes formas.

Essa obtenção de dados era feita através da comunicação do *front-end* com o *back-end*. Quando uma página web era solicitada ao *front-end*, este requisitava ao *back-end* os dados necessários para preenchê-la. Então, o *back-end* os solicitava ao banco de dados, fazendo todos os tratamentos relevantes para devolver ao *front-end* os dados já organizados de acordo com a solicitação.

Esses dados, por fim, eram colocados nas diferentes páginas e gráficos, também de

acordo com o solicitado pelo usuário, gerando uma página web que era enviada ao browser, permitindo que o usuário visualizasse a página que pediu e fizesse novas interações.

Para melhor compreensão do exposto neste capítulo, expõem-se dois diagramas de sequência, sendo o primeiro deles referente ao processo de povoar o banco de dados, pelo operador da aplicação, e o segundo referente à interação de um usuário no sistema:

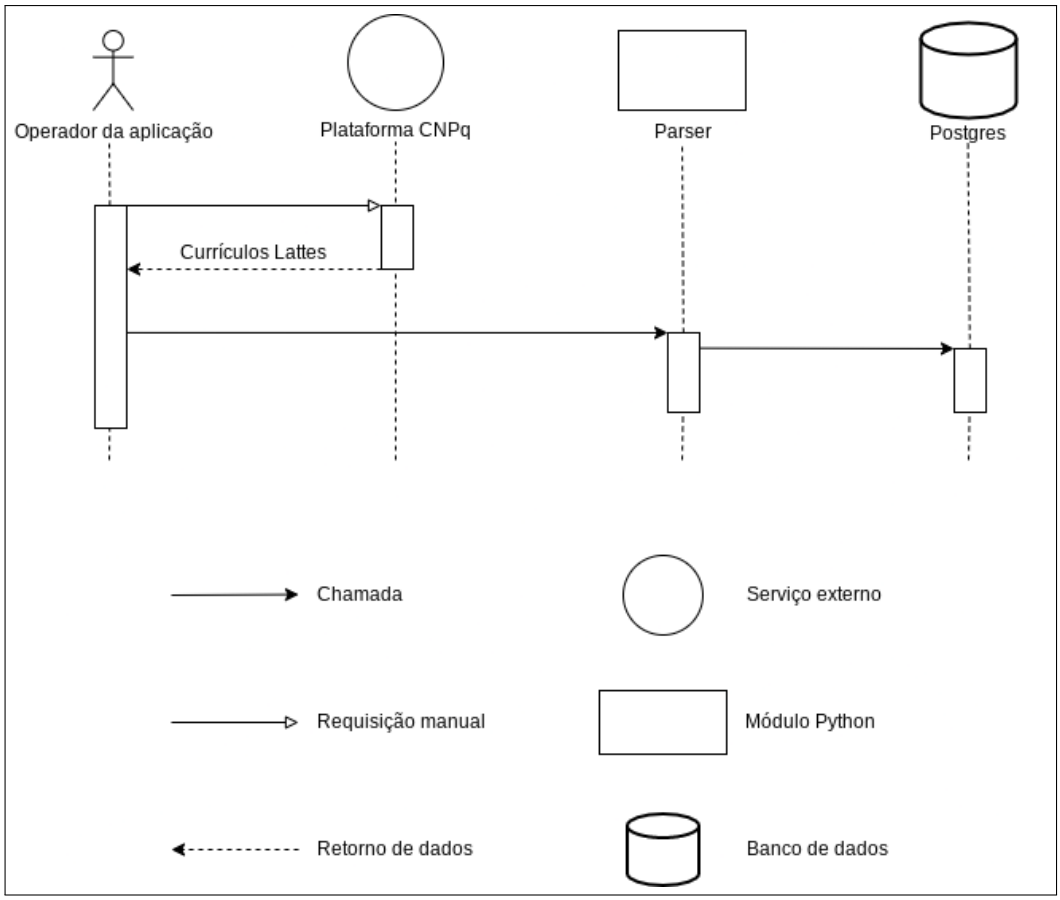

Figura 2.4: *Povoamento dos dados pelo operador da aplicação.*

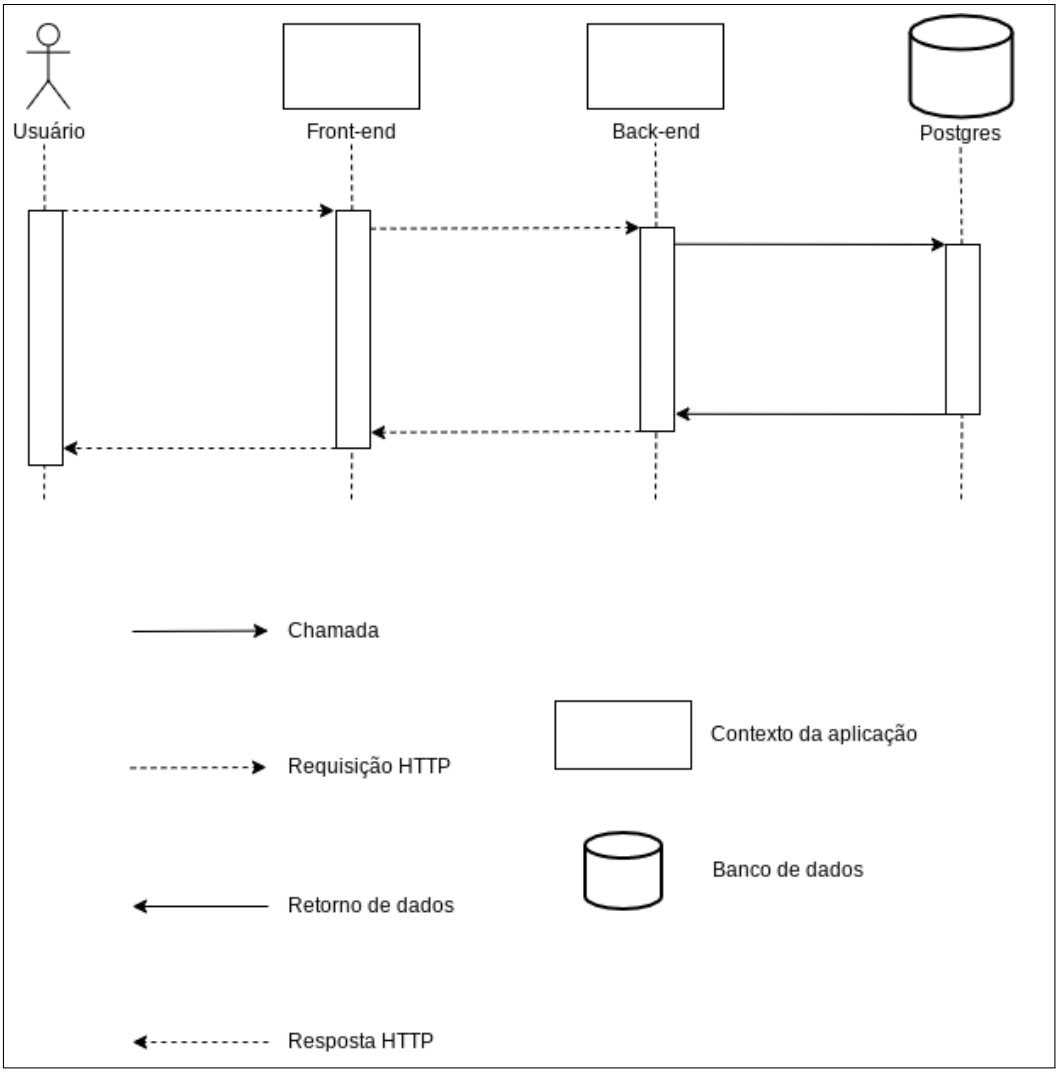

Figura 2.5: *Interação do usuário no FAU Aberta.*

2.8 Elementos complementares

Além da arquitetura do sistema em si e de seu funcionamento, há outros fatores interessantes de se citar em relação ao estado do projeto.

Primeiramente, podem ser mencionados os testes, que embora não façam parte do sistema de fato, isto é, quando este está sendo executado, permitem que o código continue funcionando após alterações.

No código do projeto FAU Aberta havia diversos testes para o *back-end*. Como mencionado anteriormente, cada app do Django possuía um conjunto de testes referentes a suas ações e rotas. Esses testes podem ser realizados diretamente através de comandos oferecidos pelo Django.

Além disso, no GitLab, havia uma *pipeline*[7] configurada que, além de realizar os testes do

[7] Ver **Apêndice A**.

back-end, também verificava que as imagens dos contêineres Docker podiam ser construídas e executadas sem erro.

Outro fator interessante, também no GitLab, era a presença de uma Wiki do sistema, contendo diversas informações úteis como: instruções para inicialização, configuração e execução do sistema; explicações de como fazer a inserção dos dados no Lattes no banco de dados; e a estrutura da API web para que se soubesse como realizar as consultas. O projeto também contava com alguns arquivos README, detalhando o funcionamento e instruções dos elementos que se encontravam no mesmo diretório que esses arquivos.

Capítulo 3

Atividades de Desenvolvimento

Esse capítulo explicita todas as atividades desenvolvidas durante a execução deste trabalho. Cada atividade é apresentada em três partes: **o problema**, apresentando o problema que se estava buscando resolver; **a solução**: exibindo a solução aplicada; e **a execução**, exibindo os detalhes técnicos e operacionais com que a solução foi colocada em prática. A seguir, apresenta-se uma tabela que exibe as atividades, separando-as em três categorias: "Código", que se refere a análises e mudanças internas do sistema, visando melhoria de legibilidade, manutenção ou desempenho do sistema, sem alterar suas funcionalidades; "Ambiente", denotando análises e mudanças nas tecnologias e ferramentas usadas; e "Nova Funcionalidade", que representa a criação e adição de novas funcionalidades e elementos do sistema.

	Código	Ambiente	Nova Funcionalidade
Análise crítica do código	×	×	
Desacoplamento do Docker		×	
Implantação na nuvem		×	
Sincronização entre o *Schema* e os *Models*	×		
Implementação do mapa nacional			×
Otimização das rotas	×		
Remoção do corsproxy	×	×	
Registro de atualizações			×
Refatoração de duplicações no *front-end*	×		
FAQ			×
Configuração para *deploy*		×	
Refatoração de duplicações no *back-end*	×		
Dados agregados	×	×	
Criação de componentes comuns	×		
Expansão para a FEA-RP			×
Automatização da coleta de dados	×		

Tabela 3.1: *Atividades desenvolvidas durante o trabalho, definindo suas categorias.*

3.1 Análise crítica do código

O problema: Antes de realizar qualquer atividade que envolvesse escrita de código ou algo relacionado, a equipe estudou o código do projeto, para entender seu funcionamento, sua estrutura etc. Além disso, foi necessário avaliar as tecnologias utilizadas no sistema, para verificar se elas seriam adequadas para os novos objetivos do projeto ou se havia outras que poderiam substituí-las com melhores resultados.

A solução: Assim, foi realizada uma análise crítica do código, de sua estrutura e das ferramentas empregadas, buscando pesar seus benefícios e desvantagens, e tentando entender se seria possível utilizá-las para expandir o projeto da forma como havia sido imaginado.

A execução: Primeiramente analisou-se a infraestrutura do código, que teria sua importância ainda mais acentuada agora que o projeto tinha como objetivo a criação de múltiplos sistemas web, e portanto, uma arquitetura bem definida e organizada seria muito importante.

Verificou-se que o Docker é muito utilizado para a conteinerização e atendia a todas as necessidades do CPqs Abertas, e, portanto, optou-se por manter essa tecnologia para o desenvolvimento. Entretanto, da forma com que a infraestrutura estava organizada, o sistema estava fortemente acoplado ao Docker, isto é, sua execução só era possível com a utilização dessa ferramenta. Por conta disso, foi decidido desfazer esse acoplamento, para trazer maior flexibilidade ao projeto, garantindo que o Docker ainda pudesse ser usado, mas que a execução do sistema de outras formas também pudesse ser realizada. Este processo será explicado em mais detalhes posteriormente.

Em seguida, a análise foi direcionada para o banco de dados, que utilizava o PostgreSQL, no qual há tabelas para representar cada entidade do sistema. Por conta do objetivo do CPqs Abertas, decidiu-se por manter essa tecnologia, dado que as entidades poderiam representar os professores e suas diferentes produções, e poderiam ser estabelecidas relações entre esses objetos.

Entretanto, durante a execução do projeto, foi notada a possibilidade de se utilizar um outro tipo de banco de dados, orientado a documentos (MongoDB), que poderia oferecer algumas melhorias ao sistema, de forma que essa tecnologia também foi testada. Essas possíveis melhorias e o processo de teste e implementação também serão abordados posteriormente.

Seguiu-se então para o *back-end*, programado em Python com o *framework* Django. Inicialmente, a equipe de desenvolvimento considerou trocá-lo por um outro *framework* Python, o Flask, por conta da familiaridade com essa tecnologia. Além disso, algumas ferramentas presentes em ambos os *frameworks*, como a integração com o banco de dados e a criação das rotas, possuíam sintaxe e estruturação mais simples no Flask.

Contudo, em análise mais aprofundada, julgou-se que o tempo e esforço necessários para migrar entre os *frameworks* não compensaria as vantagens advindas dessa troca, e

portanto decidiu-se manter o Django e mover a atenção para outros pontos na estrutura do sistema.

Após esta análise, a equipe passou a examinar o *front-end*. Decidiu-se por manter nele todas as tecnologias que estavam empregadas, isto é, a linguagem JavaScript, o ambiente Node.js e a biblioteca React. Isso se deu tanto pela familiaridade que a equipe já possuía com essas tecnologias, quanto pelo fato de elas serem muito utilizadas para construir sistemas web com páginas dinâmicas. Sendo assim, seria muito mais fácil obter suporte e resolver problemas dessas ferramentas do que de outras que não são tão usadas ou que não sejam feitas especificamente para sistemas web como o do CPqs Abertas.

A última decisão do grupo foi a de remover o contêiner corsproxy como um todo, já que continha um pequeno servidor com um único e simples objetivo, fazer uma requisição particular para o site do CNPq. Devido a seu uso muito específico, e para diminuir a complexidade do sistema reduzindo o número de contêineres, optou-se por remover este servidor completamente, já que as requisições poderiam ser feitas de forma direta, sem a necessidade de um *proxy* entre as etapas.

3.2 Desacoplamento do Docker

O problema: Como mencionado no tópico anterior, a infraestrutura foi organizada de forma que o projeto havia se tornado fortemente acoplado ao Docker. Esse acoplamento acontecia pois os diretórios do *back-end* e do *front-end* estavam organizados de forma incorreta para a execução sem utilização de contêineres. Sendo assim, uma das instruções de seus respectivos Dockerfiles era colocar os arquivos na organização correta dentro do contêiner. Ou seja, apenas com a utilização do Docker os arquivos ficavam na estrutura correta para inicialização do sistema.

Desta forma, não era possível executar o programa sem o uso dessa ferramenta, o que poderia ser impeditivo em alguns casos. Assim, o desacoplamento do Docker se apresentava como algo necessário para o prosseguimento do projeto.

A solução: Buscou-se, portanto, reestruturar os diretórios do *back-end* e do *front-end* para que ficassem em um formato que possibilitasse a execução do sistema sem o Docker, isto é, sem a necessidade das instruções utilizadas por ele para organizá-los. Outra importância dessa reestruturação era fornecer à equipe um melhor entendimento do projeto, tanto porque seria necessário entender a arquitetura existente para poder alterá-la sem prejudicar o sistema, quanto porque dessa forma todos os desenvolvedores saberiam como seria a nova organização.

A execução: Para realizar a reestruturação, os diretórios foram reorganizados, sendo movidos de forma a colocá-los na estrutura correta para a execução do sistema, sem necessidade dos comandos presentes nos Dockerfiles. Sendo assim, também foram feitas mudanças nesses arquivos, para remover as instruções que estruturavam os diretórios, já que essa tarefa não seria mais necessária.

Todo esse processo foi realizado com o auxílio do mestrando João Daniel, que atua na

área de arquitetura de software e tem bons conhecimentos do Docker. Assim, não apenas houve ajuda na reestruturação, como também a equipe pôde aprender conceitos e boas práticas na utilização dessa ferramenta.

3.3 Implantação na nuvem

O problema: Neste momento do projeto, o sistema do FAU Aberta estava implantado no servidor da FAU, pela sua equipe de tecnologia própria, a partir das instruções dos desenvolvedores anteriores. Desta forma, não seria possível para a equipe de desenvolvimento atualizar o sistema a cada modificação feita ou funcionalidade adicionada, visto que os responsáveis pelo *deploy* eram outros. Isso era particularmente ruim pois, devido às práticas ágeis do projeto, era de interesse de toda a equipe que houvesse comunicação constante entre todas as partes, tanto internas (equipes de desenvolvimento, design e coordenadores) quanto externas (clientes do projeto). Essa comunicação envolvia, entre outras questões, a visualização frequente das modificações realizadas no sistema. Portanto, era necessária uma forma de permitir que o sistema pudesse ser visualizado facilmente a qualquer momento, e que este estivesse sempre atualizado.

A solução: Para permitir a verificação regular das alterações do sistema, por parte dos membros das outras equipes e dos clientes, optou-se por fazer o *deploy* do sistema também na nuvem. Assim, os desenvolvedores poderiam atualizá-lo diretamente, sem intermédio da equipe da FAU, o que tornaria o processo muito mais ágil, permitindo uma frequência de atualização maior.

É interessante ressaltar que esse sistema seria apenas para o desenvolvimento, isto é, apenas para a equipe do projeto e os clientes da FAU. Sendo assim, não havia divulgação do sistema na nuvem, e o site oficial do FAU Aberta permanecia aquele implantado pela equipe de tecnologia da FAU.

A execução: Após a análise de diferentes serviços para realizar o processo de *deploy* na nuvem, com a ajuda do mestrando João Daniel, decidiu-se por utilizar o Heroku, para gerenciar o banco de dados e o *back-end*, e o Netlify para o *front-end*.

O Heroku é um serviço que oferece uma plataforma completa para que desenvolvedores possam hospedar seus sistemas de forma simples. Um dos benefícios que ele oferece, por exemplo, é a integração com o Git, que permite que se crie uma cópia do repositório dentro do serviço. Assim, quando se atualiza esse repositório, o sistema também é automaticamente atualizado no Heroku. Essa velocidade na atualização permite que o resultado seja observado e avaliado rapidamente. Além disso, possui simples integração com o banco de dados e um ótimo suporte a Python, e, por isso, optou-se por utilizá-lo no banco e no *back-end*.

Devido à baixa complexidade do banco de dados, no sentido de que não havia configurações avançadas, bastou informar ao Heroku que se desejava um banco gerenciado pelo PostgreSQL. Então, o serviço criou o banco de dados utilizando a nuvem da Amazon e forneceu o endereço, permitindo que se pudesse configurar o *back-end* para acessá-lo.

O *back-end* exigiu uma configuração um pouco maior. Primeiramente foi necessário criar um arquivo detalhando ao Heroku quais comandos deveriam ser executados quando o sistema estivesse no ar. Em seguida, realizou-se a integração com o Git, habilitando o serviço para ser atualizado quando o repositório também o fosse.

Já o Netlify também oferece uma plataforma como serviço, mas esta tem foco maior no *front-end*, com suporte para vários dos *frameworks* e bibliotecas mais utilizados. Deste modo, optou-se por utilizá-lo nesta parte do sistema.

Assim como o Heroku, o Netlify também oferece uma integração com o Git que possibilita atualização automática a partir de novos *commits*[1], o que seria de grande ajuda para o projeto. Sendo assim, para implantar essa parte do sistema bastou realizar a integração com o Git, e informar o endereço do *back-end* (que foi fornecido pelo Heroku), para que ele soubesse para onde deveriam ser feitas as requisições do *front-end*.

Ao final, a equipe conectou-se à máquina em que o *back-end* estava hospedado, através do Heroku, para que o *parser* fosse utilizado para povoar o banco.

Com isso, o sistema estava completamente implantado na nuvem, permitindo acesso rápido para todos os membros da equipe e clientes da FAU, e também possibilitando atualizações frequentes a partir das modificações realizadas.

3.4 Sincronização entre o *Schema* e os *Models*

O problema: Ao analisar mais a fundo o *back-end*, percebeu que os *models*, isto é, os modelos que representam as diferentes entidades do banco de dados, estavam desatualizados. Por conta disso, investigou-se de onde vinham os dados que garantiam que, no banco, as entidades estivessem todas corretas.

Percebeu-se então que, no diretório do banco de dados, havia um arquivo SQL para determinar instruções para a criação do banco de dados e suas tabelas. Esse arquivo era utilizado pelo contêiner do banco de dados para criá-lo com todas essas configurações. Sendo assim, o arquivo do Django detalhando todos os modelos não era utilizado na geração do banco de dados, apenas durante a execução do sistema para o mapeamento objeto-relacional.

Isso não apenas causou confusão na equipe, em um primeiro momento, como também poderia afetar futuras equipes do projeto. Isso porque, não apenas trazia dúvidas sobre quem era responsável por determinar os modelos, como também fazia com que alterações nos modelos pelo Django não fossem refletidas no sistema. Dessa forma, quaisquer alterações deveriam ser feitas diretamente no arquivo SQL, o que não é usual e também aumenta a chance de erros.

A solução: Para evitar as confusões causadas por essa estruturação, determinou-se que o ideal seria centralizar a responsabilidade da definição dos modelos no Django. Assim, removeria-se a necessidade de utilização do arquivo SQL e garantiria-se que todas as

[1] Ver **Apêndice A**.

mudanças ou adições nos modelos do Django fossem automaticamente refletidas no banco de dados.

A execução: Para isso, utilizou-se o arquivo SQL para verificar todas as diferenças entre o banco atual e correto (no arquivo SQL) em relação ao banco desatualizado (no Django), e então adaptar o arquivo do Django para que ficasse igual. Tendo isso feito, reconfigurou-se o Docker para que agora o Django criasse o banco de dados de acordo com seus modelos. Dessa forma, o Django ficaria responsável pela estruturação do banco de dados, como pretendido.

3.5 Implementação do mapa nacional

O problema: O site do FAU Aberta possuía como uma das formas de visualização dos seus dados um gráfico com o mapa-múndi, agrupando os dados de acordo com o país de origem de cada produção. Entretanto, sendo a FAU parte de uma universidade brasileira, e o sistema FAU Aberta algo voltado principalmente para a população do país, seus membros sentiam falta de um mapa nacional, que funcionasse de forma similar, mas realizando o agrupamento por estado.

A solução: Para realizar essa implementação, decidiu-se adicionar informações sobre os estados das produções no banco de dados, visto que o *parser* não extraía esse dado, e então basear-se no mapa internacional para criação do nacional.

A execução: Como mencionado, o primeiro passo para implementar o mapa nacional foi o de coletar os dados sobre os estados das produções através dos currículos Lattes. Realizar essa tarefa, porém, apresentou-se com um desafio a mais, uma vez que no currículo Lattes, as produções não possuem um campo para informar o estado em que foram feitas, mas apenas para a cidade. Desta forma, seria necessário primeiro extrair os estados a partir das cidades encontradas. Contudo, notou-se que as informações inseridas nos campos sobre cidades não estavam padronizadas, com algumas produções contendo o estado, outras a cidade e algumas com ambos, além de algumas com erros ortográficos.

A solução encontrada para resolver essa questão foi adquirir um arquivo que possuísse todas as cidades brasileiras e as mapeasse para seus respectivos estados. Então, adicionou-se ao *parser* um processamento que coletaria a cidade de cada produção e utilizaria esse mapeamento para guardar o estado correto no banco. Este processamento também armazenaria em um *log*[2] quaisquer cidades que não pudessem ser mapeadas (em casos de cidades de outros países ou grafadas incorretamente), para que os professores pudessem ser informados sobre possíveis erros de preenchimento.

Em seguida, para inserir a funcionalidade de exibição do mapa nacional de fato, tomou-se como base o código utilizado na criação do mapa internacional, tanto do *back-end* quanto do *front-end*. Isso foi feito pois a forma de agrupar os dados e exibi-los seria igual,

[2] Ver **Apêndice A**.

com exceção do mapa utilizado.

Tendo observado a implementação do mapa internacional então, restou apenas encontrar um arquivo que definisse corretamente o mapa do Brasil para permitir sua exibição no site. Além disso, seria necessário garantir uma forma de que os dados fossem representados nas posições corretas em relação a cada estado.

Verificou-se que as bibliotecas utilizadas no mapa internacional para esse tipo de funcionalidade também poderiam ser usadas para o nacional. Entretanto, notou-se que os dados eram exibidos no centro de cada estado, e não sobre suas capitais, como é usual em mapas desse tipo.

Assim, buscou-se usar as informações de latitude e longitude de cada estado, também com as bibliotecas citadas anteriormente, garantindo-se que os dados seriam exibidos nas devidas posições.

É relevante ressaltar que esse tipo de gráfico seria mostrado em diferentes partes do sistema, como na página principal, nas páginas de departamento, e nas diferentes produções. Contudo, da forma em que estava estruturado, o *back-end* possuía rotas diferentes para cada uma dessas seções, e, portanto, seria necessário modificar todas elas.

Por conta disso, decidiu-se implementar a nova funcionalidade em apenas uma seção, neste caso, a produção bibliográfica, para realizar todos os testes, mostrar para os clientes e organizar quaisquer outros pontos. Só depois de todas essas etapas a funcionalidade foi replicada para todas as seções e, então, testada e exibida para a equipe. Com a aprovação por parte dos membros da FAU, dirigiu-se às próximas atividades.

3.6 Otimização das rotas

O problema: Durante os testes realizados para o mapa nacional, percebeu-se, acessando o site através do Netlify, que alguns elementos das páginas estavam levando muito tempo para carregar. Verificou-se que as requisições que consistiam de contagens de dados a partir de certos parâmetros estavam levando mais de 10 segundos para retornar uma resposta. É fácil notar que uma espera dessa magnitude impacta fortemente o usuário, que deve aguardar todo esse tempo para visualizar um certo dado.

Visto que localmente esse problema não existia, supôs-se que a demora deveria estar sendo causada pela comunicação entre os serviços do sistema, isto é, entre o *front-end* e o *back-end*, ou entre este e o banco de dados.

Como o *front-end* simplesmente requisitava um tipo de dado e este seria devolvido pelo *back-end*, parecia estranho que esse fosse o motivo do atraso, dada a baixa complexidade. Sendo assim, buscou-se investigar o tratamento dos dados realizado pelo *back-end*, pois talvez esse não estivesse otimizado.

Notou-se então que, da forma com que as rotas estavam implementadas, as funções de contagem, que agrupavam os dados por diferentes categorias (departamentos, tipos de produção etc), estavam muito ineficientes.

Isso acontecia porque, ao invés de pegar os dados de uma única vez no banco e então

realizar todos os processamentos necessários, a função buscava fazer uma requisição que já fornecesse os dados totalmente processados. Da maneira com que estava programada, entretanto, ela fazia isso executando milhares de consultas para o banco de dados. Com o sistema na nuvem, e o banco de dados não estando hospedado na mesma máquina que o *back-end*, esse processo demorava muito, já que cada uma dessas consultas precisava viajar pela rede entre um serviço e outro.

A solução: Por conta disso, decidiu-se refatorar completamente essas rotas, reduzindo ao máximo o número de consultas, culminando em um grande aumento em sua eficiência. Para realizar essa tarefa, não era interessante também fazer uma requisição básica que solicitasse todos os dados e deixasse que o Python fizesse todo o processamento, pois a complexidade de algumas operações seria muito grande.

A execução: Desta forma, buscou-se fazer as consultas ao banco de dados de forma bem arquitetada para que os dados já viessem agrupados de certas formas, evitando que o Python precisasse fazer esse agrupamento.

Neste ponto do trabalho a equipe encontrou uma grande dificuldade em trabalhar com o ORM do Django: realizar os agrupamentos. Em SQL, é possível utilizar a cláusula *GROUP BY* para agrupar as entidades de acordo com um ou mais atributos, realizando operações a partir desse agrupamento. No processamento em questão, essa operação seria justamente a contagem, que iria agrupar as produções de acordo com um determinado parâmetro.

No Django, entretanto, implementar esse tipo de operação não é tão simples. O ORM não possui uma função para fazer o *GROUP BY* diretamente, e portanto deve-se construir a cláusula de outra forma. Entretanto, devido à documentação dessa operação no Django não ser muito clara, entender como criar essa cláusula não é simples. Além disso, havia detalhes sobre o comportamento das funções de ORM que o Django não deixava explícitos, os quais foram descobertos após múltiplos testes e tentativas de execução.

Passado esse processo, com as consultas já funcionando da forma pretendida, bastou realizar um pequeno processamento no Python para formatar os dados da forma que o *front-end* os esperava. Esse método de refatoração foi aplicado em todas as rotas que trabalhavam com contagem e em que foi verificado um número extremamente grande de consultas.

Ao fim de toda a refatoração, todas as rotas que apresentavam esse problema passaram a levar entre meio e um segundo, tempo muito mais aceitável para que um usuário espere o carregamento.

3.7 Remoção do corsproxy

O problema: Como exposto anteriormente, o sistema do FAU Aberta apresentava um contêiner denominado corsproxy, que tinha como utilidade executar um servidor *proxy* para permitir que o *front-end* fizesse requisições para o site do CNPq Lattes. Como só havia uma requisição que precisava ser feita, esse servidor era muito pouco utilizado e não era realmente necessário, visto que a requisição podia ser feita diretamente. Assim, o

sistema ficava mais complexo, devido a existência de um contêiner a mais, sem nenhum tipo de benefício.

A solução: Percebeu-se que bastaria remover o contêiner corsproxy, visto que este não era realmente necessário, realizando pequenos ajustes nas partes do sistema que o utilizavam.

A execução: Para isso, primeiro foram refatoradas as partes do *front-end* que faziam as requisições para o servidor *proxy*, fazendo com que estas fossem feitas diretamente ao site do CNPq. Em seguida, já se pôde apagar completamente o contêiner, e com alguns testes detectou-se que o sistema continuava funcionando normalmente, agora com complexidade levemente reduzida.

3.8 Registro de atualizações

O problema: O sistema do FAU Aberta não era atualizado com os dados do Lattes em tempo real, visto que era necessário baixar os currículos manualmente e então executar o *parser*. Por conta disso, as atualizações eram apenas periódicas, de forma que o sistema poderia não conter alguma produção adicionada por um docente em seu currículo. Visto que o objetivo do site é o de divulgar as produções, algum docente poderia se incomodar com a ausência da produção no site, e portanto era necessária uma forma de se verificar quando os dados foram atualizados.

A solução: Devido a isso, decidiu-se exibir no canto inferior direito do site a data da última atualização dos dados no sistema. Dessa forma, os usuários verificariam a data de atualização e não estranhariam ausências de produções mais recentes, por exemplo.

A execução: Para isso, bastou apenas armazenar a data de atualização no banco de dados. Assim, configurou-se o *parser* para que, sempre que terminasse sua execução com sucesso, atualizasse essa data. Por fim, estabeleceu-se uma rota no *back-end* que retorna essa data, permitindo que o *front-end* a obtenha e a exiba.

3.9 Refatoração de duplicações no *front-end*

O problema: O sistema do FAU Aberta apresentava seis tipos diferentes de produções acadêmicas: prêmios e títulos, orientações, bancas, produções artísticas, bibliográficas e técnicas. Embora essas produções tenham diferenças conceituais e entre seus atributos, suas visualizações no sistema eram muito parecidas. Isto é, os gráficos, seus *layouts* e a ordem em que eram apresentados eram os mesmos, independentemente do tipo de produção.

Entretanto, no código-fonte, cada uma das páginas que apresentava essas produções possuía um componente distinto, mas estes eram praticamente iguais, visto que as páginas

eram bastante similares. Sabe-se no entanto que essa não é uma boa prática, visto que a manutenção torna-se extremamente difícil. Isso porque, ao fazer algum tipo de alteração, esta deve ser replicada nos seis arquivos, o que consome tempo e aumenta as chances de erro.

A solução: Decidiu-se então por analisar o código-fonte em busca de pontos em comum entre todos os arquivos, para que estes fossem extraídos a um arquivo compartilhado, de forma que uma produção tivesse em seu código apenas o que era específico a ela. Desta maneira, qualquer alteração futura em aspectos gerais poderia ser feita apenas uma vez, neste arquivo comum, facilitando essas modificações.

No React, ferramenta usada no *front-end*, este processo não apenas é simples, como também é um de seus pontos fortes. Sua estratégia de definir os elementos do sistema em componentes, que podem ser instanciados com diferentes parâmetros de forma dinâmica, permite essa generalização.

A execução: Sendo assim, criou-se um componente comum para todas as produções, contendo o formato da página, os gráficos a serem exibidos etc, e então, para cada produção, criou-se um arquivo bem mais simples, que utiliza esse componente comum, apenas instanciando-o com diferentes parâmetros.

Com isso, obtém-se maior manutenibilidade e também melhor legibilidade do código e da estrutura do sistema, ao remover diversos arquivos que possuem muita repetição de código.

3.10 FAQ

O problema: Em um projeto como o CPqs Abertas, de divulgação de dados e apresentação de informações, é importante informar os usuários sobre detalhes como sua origem, frequência de atualização, etc. Sendo um projeto de âmbito acadêmico, também é interessante mostrar quais os institutos participantes, o intuito do projeto e ainda quaisquer outras dúvidas que os usuários possam ter. Mostra-se necessária então uma forma de exibir essas explicações e permitir que usuários tirem suas dúvidas.

A solução: Por conta disso, optou-se por implantar no sistema uma seção de FAQ (*Frequent Asked Questions*), que além de exibir perguntas sobre informações importantes do projeto, também possibilita o envio de perguntas para a equipe.

A execução: Até este momento, a maior parte das atividades desenvolvidas havia sido mais focada em aspectos mais internos do sistema, isto é, questões tecnológicas e de desenvolvimento, como otimizações e refatorações. Já o FAQ se tratava de algo mais externo, que seria consumido diretamente pelo usuário e visualizado no site.

Desta forma, a implementação do FAQ exigiu maior colaboração entre as equipes de design e desenvolvimento. Enquanto a primeira buscou criar um estilo para o FAQ

que conversasse com o resto do sistema, a segunda ficou responsável por implementar a funcionalidade seguindo o estilo proposto. Para isso foi necessária bastante comunicação, para garantir que os detalhes decididos pela equipe de design pudessem ser inseridos no sistema e para que a equipe de desenvolvimento o fizesse da forma correta.

3.11 Apresentação para a FAU e início do FEA-RP Aberta

Nesta etapa do projeto, foi realizada a apresentação do sistema para a FAU, em uma reunião que contou com a equipe do projeto, a professora da FAU Beatriz Bueno, uma das idealizadoras do projeto, e as professoras Ana Lucia Duarte Lanna e Mônica Junqueira de Camargo, também da FAU.

Nesta reunião foram apresentadas todas as modificações feitas no sistema e então foi feita sua entrega, confirmando que o novo sistema seria divulgado no site da FAU e que a equipe poderia se dirigir agora a um novo instituto. O site FAU Aberta pode ser acessado em http://fauaberta.fau.usp.br:3000.

Após esta reunião, com a aprovação do sistema pela FAU, o novo foco do projeto tornou-se a FEA-RP, que já havia sido escolhida previamente como o próximo instituto. Uma reunião com a equipe da FEA-RP foi então realizada, para mostrar como era o sistema da FAU, para que pudessem ter uma referência, e indagar se eles realmente possuíam interesse em um sistema semelhante e como ele seria.

Enquanto a posição oficial da FEA-RP era aguardada, a equipe de desenvolvimento buscou preparar todo o sistema e sua estrutura para a inserção de uma nova unidade. Devido ao maior volume de tarefas que seria feito a partir desse momento, a equipe optou por parar a rotina de três a quatro sessões de *Mob Programming* semanais, passando para uma reunião semanal para divisão de tarefas e subsequente desenvolvimento individual.

3.12 Configuração para *deploy*

O problema: Dada a aprovação do sistema pela FAU, o próximo passo era repassar todo o código à sua equipe de tecnologia para que ele pudesse ser executado em seus servidores. Entretanto, a inicialização não era imediata, devido ao povoamento do banco com os Lattes, por exemplo. Sendo assim, era necessário estruturar bem os passos da inicialização para que a equipe da FAU pudesse executar o sistema corretamente.

Além disso, por conta da expansão do sistema para outras unidades, outras equipes eventualmente teriam que realizar esses passos em seus respectivos servidores. Desta maneira, uma forma mais automatizada para a execução era interessante.

A solução: Decidiu-se então por criar um novo *script*[3] de *setup*[4], a partir de outro

[3] Ver **Apêndice A**.

[4] Ver **Apêndice A**.

criado pela equipe anterior, o qual pudesse ser executado apenas uma vez e já seria responsável por povoar o banco de dados e colocar o sistema no ar. Além disso, algumas instruções para configuração do sistema antes de sua execução deveriam ser passadas para a equipe.

A execução: Utilizando o Bash, o *script* foi criado, a partir dos comandos que eram empregados para inicializar o sistema, desde o povoamento do banco de dados até a execução em si. Este *script* pode ser visualizado no **Apêndice B**.

Em seguida, foram definidas as instruções de configuração do sistema para que esse *script* pudesse ser executado corretamente. Entre essas instruções estavam: pré-requisitos, como a instalação do Docker; variáveis de ambiente necessárias, tanto do *back-end* quanto do *front-end*; e informações sobre o procedimento correto para inserção dos currículos no sistema, isto é, em que diretórios colocar os arquivos XML dos currículos para que o banco de dados fosse povoado corretamente. Essas orientações foram também registradas na Wiki do projeto, acessível pelo GitLab.

Como mencionado, eventualmente esse processo seria realizado por equipes de tecnologia de outros institutos que aderissem ao projeto. Por conta disso, tem-se que em algum momento desse processo deveria haver algo que distinguisse o instituto para o qual o sistema estava sendo executado. Do contrário, todos os sistemas seriam exatamente iguais.

Sendo assim, aproveitou-se a oportunidade para iniciar a preparação da estrutura do projeto para lidar com múltiplos institutos. Para isso, primeiro foram identificados quais elementos do sistema eram independentes de instituto e quais não eram. Esses últimos então foram separados em diretórios específicos da unidade a qual se referiam. A partir disso, com uma variável de ambiente, definida na etapa de configuração, o sistema é informado sobre o instituto que deve executar, utilizando os diretórios respectivos para empregar os elementos corretos.

3.13 Refatoração de duplicações no *back-end*

O problema: Em vista da futura expansão do sistema para inserção do site da FEA-RP, era necessário fazer certas preparações no *back-end*. Isso porque, sendo este o pedaço do sistema que possui as regras de negócio, era preciso permitir que houvesse uma distinção clara entre as operações da FAU e da FEA-RP.

Para isso, o sistema do *back-end* deveria ser organizado de tal forma que tanto o *front-end* da FAU quanto o da FEA-RP pudessem usá-lo, sem que houvesse conflito nas rotas ou nas operações.

Entretanto, antes de fazer isso, outro passo era necessário: a refatoração das operações duplicadas no *back-end*. Isso porque, assim como no *front-end*, cada um dos seis tipos de produção possuía diversas rotas que, essencialmente, realizavam a mesma operação, principalmente nas referentes a contagem. Sendo assim, quando fossem realizadas as preparações do *back-end* para lidar com múltiplos institutos, qualquer mudança deveria ser aplicada em seis lugares diferentes, o que atrasaria o desenvolvimento e aumentaria a

chance de erro.

A solução: Do mesmo modo que no *front-end*, deliberou-se por identificar todas as operações que eram comuns aos diferentes tipos de produção, e transformá-las em funções únicas que seriam compartilhadas. Assim, as rotas de cada uma das produções apenas forneceriam os parâmetros para essa função comum, mas toda a lógica estaria armazenada em um único lugar. Com isso, quaisquer alterações advindas da preparação do sistema para expansão que afetassem essas operações só precisariam ser aplicadas uma vez.

A execução: Antes de remover essas duplicações, no entanto, foi necessária uma refatoração nos modelos referentes às entidades do banco de dados. Isso porque, da maneira com que o FAU Aberta havia sido projetado, os atributos de cada entidade tinham prefixos indicando a entidade que representavam. Por exemplo, o ano de uma produção bibliográfica possuía o nome "bib_ano", enquanto o de uma banca possuía o nome "ban_ano".

Por conta disso, generalizar as operações seria uma tarefa muito mais complicada, pois seria necessário construir o nome do atributo a partir da entidade que se estava analisando, ao invés de simplesmente usar "ano", como no caso do exemplo. Além disso, a estruturação de ORM do Django faz com que os nomes de atributos sejam usados de forma direta em alguns casos, isto é, não em uma *string* com o nome, envolta por aspas, mas sim como atributos de uma classe ou parâmetros de uma função. Sendo assim, não seria possível a construção dos nomes dos atributos de forma simples.

Depois dessa tarefa, com as entidades sem prefixos nos nomes de seus atributos, foi possível realizar a generalização das operações. Desta forma, criou-se um arquivo comum com todas elas, fazendo quaisquer ajustes necessários, e então as rotas foram configuradas para fornecer os parâmetros corretos a partir das produções a que se referiam.

Dessa maneira, permitiu-se que a reestruturação posterior do sistema fosse extremamente facilitada, o que pôde ser verificado rapidamente na implementação de uma estrutura para armazenamento dos dados agregados, que será exposta a seguir.

3.14 Dados agregados

O problema: Desde a identificação da demora de algumas rotas, mencionada no **tópico 3.6**, uma alteração do sistema estava sendo analisada pela equipe de desenvolvimento: a criação de um banco de dados agregados. Para entender o porquê dessa alteração, é necessário compreender como o sistema funciona.

Primeiramente, tem-se que a atualização dos dados é feita de forma mensal, isto é, aproximadamente uma vez por mês os currículos são baixados e seus dados inseridos no banco. Desta forma, entre atualizações, o sistema permanece com os exatos mesmos dados por um mês. Sendo assim, durante esse período, todas as operações realizadas pelas rotas terão exatamente os mesmos resultados, visto que essas são determinísticas e os dados são os mesmos.

Por conta disso, seria possível organizar o sistema para armazenar o resultado dessas

operações, de forma que não fosse necessário executá-las toda vez que fossem requisitadas. Assim, bastaria agregar os dados previamente segundo essas operações, de forma que as rotas apenas os devolvessem já tratados. Isso traria um ótimo ganho de performance, tanto pela diminuição do tempo de execução das rotas, quanto pelo fato de que apenas uma consulta ao banco de dados seria necessária, independentemente da operação.

A solução: Antes de se aplicar uma solução de fato, notou-se que não havia necessidade de empregá-la para todas as operações do sistema, mas sim para aquelas que possuíam um maior tempo de retorno, que eram as de contagem. Além disso, por não possuírem filtros muito específicos, estas rotas eram requisitadas frequentemente pelo *front-end*, e portanto sua agregação seria vantajosa.

Então, decidiu-se que essas operações seriam realizadas apenas uma vez, logo após a atualização dos dados do sistema. Os dados agregados gerados por essas operações seriam então armazenados separadamente, em um outro banco de dados, de forma que pudessem ser retornados pelas rotas diretamente, sem nenhum tipo de processamento.

A execução: O sistema gerenciador de banco de dados escolhido para essa tarefa foi o MongoDB. Nele são utilizados bancos de dados orientados a documentos, diferentemente do PostgreSQL, que é relacional. Esse tipo de banco emprega documentos, que são estruturas que funcionam usando um sistema de chave-valor. Isto é, o documento é um objeto que possui diversos pares chave-valor, os quais utiliza para armazenar seus dados.

Em contraste com os bancos relacionais, bancos orientados a documentos não necessitam de uma estrutura completamente definida para os dados que armazenam. Em um banco relacional, uma relação é uma tabela que possui diversos atributos. Embora não seja obrigatório que todos os atributos de uma relação tenham valor, estes ainda estarão representados no objeto, embora com valor nulo. Sendo assim, no caso de um novo objeto ser inserido, e este possuir um atributo novo, é necessário defini-lo para toda a tabela, e assimilar um valor (ou deixá-lo nulo) para todos os objetos já inseridos no banco de dados. Já no banco orientado a documentos, não é assim que esse processo acontece. Não é necessário definir uma estrutura explícita, informando previamente quais os campos que o objeto terá (nesse caso, as chaves). Isto é, a qualquer momento, pode-se adicionar um novo documento a uma coleção (que seria o equivalente a uma tabela no modelo relacional) que contenha novos atributos, e o sistema continuará funcionando normalmente, sem nenhuma alteração necessária.

Esse tipo de funcionamento é interessante, pois, visto que cada operação retorna um objeto com diferentes valores, não é necessário definir todas as suas estruturas na criação do banco de dados utilizado pelo MongoDB. Além disso, se surgir uma nova operação, que tenha outro formato de dado, este pode ser inserido diretamente no banco de dados sem exigir mudanças. Dado que cada operação do sistema possuía uma estrutura diferente, essa característica foi fundamental, pois permitiu que não fossem necessários ajustes nas operações do *back-end* para padronizar o formato dos dados, e nem no *front-end* para utilizar os novos formatos.

Outro fator importante é que esse formato de documentos é muito similar ao utilizado

em objetos JSON, que é um formato de dados fortemente empregado em sistemas web, que também trabalha com pares chave-valor. Sendo assim, os dados armazenados podem ser retornados para o *front-end* diretamente, isto é, sem nenhum processamento por parte do *back-end*, pois já estão no formato correto. No caso do PostgreSQL, os dados obtidos precisavam ser reformatados para um objeto chave-valor (no caso do Python, dicionários), e então enviados para o *front-end*. Com o MongoDB, isso pode ser feito imediatamente.

Contudo, não há apenas benefícios em utilizar o MongoDB. Como desvantagem têm-se, por exemplo, a necessidade de um novo contêiner, a configuração de um novo banco de dados e, obviamente, a inserção de um novo serviço no sistema. Isso aumenta os recursos utilizados para se executar o sistema, sua complexidade e, ainda, tanto o tempo de configuração do sistema quanto de seu aprendizado e entendimento por parte das equipes de tecnologia e de eventuais novos desenvolvedores do projeto.

Portanto, estabeleceu-se que deveriam ser realizados testes para verificar se os ganhos fornecidos pelo MongoDB compensariam essas desvantagens. Nesses testes, o objetivo era comparar a velocidade de resposta das rotas no caso do PostgreSQL e no do MongoDB. Isto é, comparar a velocidade de se realizar as operações quando a rota é chamada, usando os dados do banco relacional, com a velocidade de se retornar o resultado diretamente quando a rota é chamada, através do armazenado no banco orientado a documentos.

Os testes se iniciaram através do próprio navegador, a partir das ferramentas de desenvolvedor, que permitem verificar o tempo entre a requisição realizada para o *back-end* e a resposta retornada. Entretanto, julgou-se que esse método não apenas levaria muito tempo, como também não seria muito adequado, já que reproduzir os testes diversas vezes seria ainda mais demorado.

Sendo assim, a abordagem adotada foi a de utilizar o Curl para fazer os testes. O Curl é uma ferramenta de linha de comando para transferência de dados via URL, suportando múltiplos protocolos. Com ele, utilizando um simples comando é possível realizar uma requisição HTTP para o sistema e receber a resposta. Assim, bastou elencar todas as rotas que foram modificadas e então realizar 100 requisições para cada uma delas, calculando o tempo médio entre a requisição e o retorno. O *script* utilizado para esses testes encontra-se no **Apêndice B**.

A seguir apresenta-se um gráfico com os resultados obtidos após a realização dos testes:

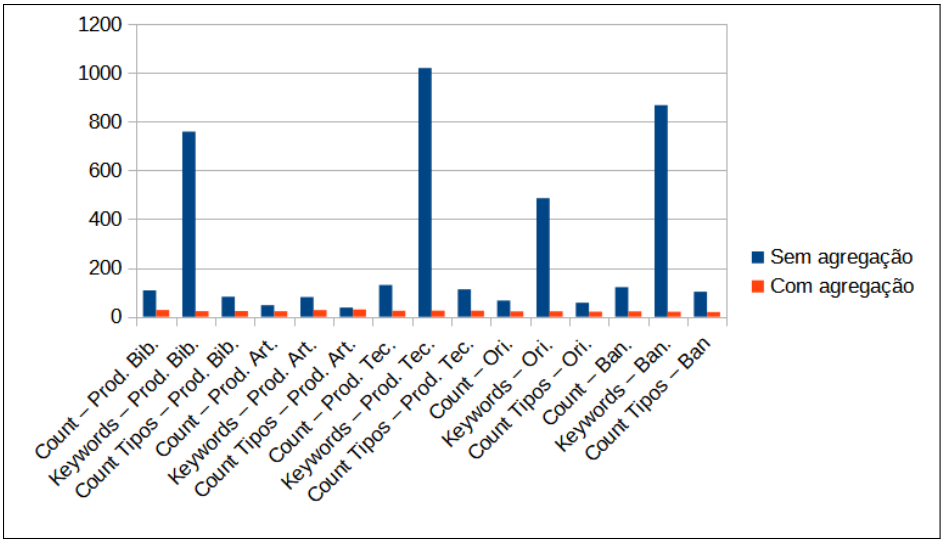

Figura 3.1: *Comparação do tempo de resposta (em milissegundos) entre o PostgreSQL (sem agregação) e o MongoDB (com agregação) em diferentes rotas.*

Comparando os resultados obtidos, observou-se uma grande melhora nos tempos de resposta quando utilizado o MongoDB. A melhora realmente era esperada, visto que todo o tempo de processamento das operações era economizado, dado que o banco de dados já possuía os dados agregados.

Por conta disso, optou-se por incluir de fato o MongoDB no sistema, pois acreditou-se que esse grande ganho de eficiência justificaria a adição de um serviço, e consequentemente, de um contêiner, no sistema.

Desta forma, foi criado um arquivo que possuía todas as operações do sistema que seriam migradas para o MongoDB. Então, configurou-se o *parser* que povoava o banco de dados para não só preencher o banco relacional com os dados dos currículos, mas também realizar todas essas operações definidas no arquivo e inserir seus resultados no banco de dados agregados. Após isso, as rotas do sistema foram configuradas para solicitar ao MongoDB o resultado da operação a que a rota se refere, retornando-o diretamente.

É importante ressaltar que, como mencionado anteriormente, a refatoração das duplicações nas operações do *back-end* teve sua eficiência comprovada neste momento. Isso porque, com as operações funcionando para todos os tipos de produção, bastava executá-las seis vezes, passando as diferentes produções como parâmetro. Isso reduziu o tamanho do arquivo com todas as operações consideravelmente, facilitando o entendimento e a legibilidade, e também a futura manutenção, devido ao código compartilhado. Caso esse processo não tivesse sido feito, o arquivo de operações precisaria conter seis funções para cada tipo de operação, uma para cada produção, tornando-o muito extenso e aumentando a chance de algum erro na programação.

Desta forma, o banco de dados agregados foi implementado, acelerando o tempo de resposta de muitas rotas, aumentando assim a eficiência do sistema.

3.15 Criação de componentes comuns

O problema: O objetivo do projeto CPqs Abertas sempre foi o de expandir rapidamente o sistema para outras unidades quando necessário, o que implica que o sistema não deve ser feito do zero para cada instituto, e que todos os pontos comuns devem ser aproveitados ao máximo.

No *front-end* essa tarefa se torna ainda mais interessante, visto que é a parte do sistema em que esse processo seria aproveitado ao máximo. Isso acontece porque é a parte que tem contato direto com o usuário, e portanto, a que mais se diferencia entre institutos. Entretanto, criar cada *front-end* do começo não seria muito eficaz, não apenas pelo tempo exigido (que retiraria a possibilidade de expansão rápida), mas também pois é fácil notar que, embora diferentes, esses sistemas inevitavelmente teriam partes comuns.

Sendo assim, era necessária uma maneira de organizar todos esses elementos comuns em um único lugar, de forma que a criação de um novo *front-end* fosse extremamente facilitada, utilizando-se desses elementos.

A solução: Para realizar essa organização, decidiu-se por elencar todos os componentes e elementos que seriam comuns entre os sistemas de diferentes unidades e então criar os componentes gerais que poderiam ser usados em múltiplos sistemas.

A execução: A habilidade do React de criar componentes apresentou-se então como uma ferramenta muito poderosa. A partir dela, seria possível definir componentes gerais a partir das características comuns dos sistemas, os quais poderiam ser utilizados por todos os sistemas em que fosse necessário.

O primeiro passo foi, como mencionado, avaliar quais os componentes e elementos que poderiam ser utilizados em diferentes sistemas da mesma maneira. Essa tarefa foi realizada com cautela, pois os únicos institutos que se tinha como referência eram a FAU e a FEA-RP. Essa última, entretanto, havia solicitado um sistema igual ao da FAU em termos de organização das páginas e de seus *layouts*, diferindo-se apenas na identidade visual. Por conta disso, seria fácil pensar em todos os componentes do sistema como comuns, visto que ambos os institutos os possuíam. Contudo, era necessário deliberar se esses componentes funcionariam como comuns no caso de um terceiro instituto solicitar *layouts* ou outros elementos distintos dos outros dois.

Feita essa identificação, partiu-se para a criação dos componentes comuns de acordo com o que havia sido verificado durante a etapa anterior. Para isso, dentro do diretório referente ao *front-end*, foram criados um diretório "common", para que esses componentes fossem colocados, e diretórios para os diferentes institutos, em que ficariam os componentes específicos de cada unidade.

Então, os elementos de cada instituto poderiam importar os componentes do diretório comum, e portanto, não haveria necessidade de criá-los de novo. Um exemplo deste tipo de componente é o gráfico de barras com as produções separadas por tipo. Para cada instituto e cada produção, os dados que populam o gráfico, suas cores e seus tipos são diferentes, mas a estrutura básica do gráfico permanece a mesma. Sendo assim, ele pôde ser colocado

como um componente comum, e cada instituto apenas o instancia com os parâmetros corretos, sem a necessidade de criá-lo novamente.

Além disso, esses componentes podem ser encadeados, de forma que um componente comum seja utilizado por um outro componente comum, não tão geral, que então será usado pelos componentes específicos. A ideia com isso, é, portanto, que o sistema possa ser "montado". Isso é, a partir das solicitações de um instituto, basta selecionar os componentes comuns que se deseja e adicioná-los da forma que for necessária, utilizando os parâmetros corretos para sua instanciação. Assim, o tempo de desenvolvimento se acelera muito, pois basta configurar os parâmetros específicos de cada componente, sem recriá-los.

Esse encadeamento também torna eventuais manutenções e alterações no sistema mais fáceis. Caso algum instituto solicite um componente que possui apenas uma versão um pouco mais específica entre os comuns, basta extrair a parte mais geral para um novo componente e utilizá-lo. Assim, os sistemas que utilizam aquele componente não são afetados, e o novo pode ser criado sem empecilhos. Também é muito simples adicionar um componente comum, visto que isso não interferirá em nenhum outro sistema. Dessa forma, conforme a entrada de outros institutos no projeto, a quantidade de componentes do sistema ficará cada vez maior, o que por sua vez acelerará ainda mais o desenvolvimento de sistemas para novos institutos.

Com a modularização do sistema pronta, e todos os componentes da FAU devidamente configurados para utilizar os elementos comuns criados, restava apenas definir uma maneira de estabelecer qual instituto está sendo executado. Como mencionado anteriormente, o sistema já possuía uma variável de ambiente configurada antes de sua execução, que era responsável por definir o instituto. A partir dela, o *front-end* se dirige ao diretório respectivo àquele instituto, localiza o arquivo App.js, que é o componente principal do sistema, e o carrega, iniciando, assim, o sistema específico daquele instituto.

3.16 Expansão para a FEA-RP

O problema: Com a refatoração do *back-end* finalizada, tanto na eliminação de duplicações quanto na criação do banco de dados agregados, e a modularização do *front-end* também concluída, era chegada a hora de expandir o sistema para a FEA-RP.

A solução: Para realizar a expansão deveria-se atuar em todas as partes do sistema. Em relação ao banco de dados, era necessário adquirir os currículos Lattes dos docentes da FEA-RP para o povoamento. Já no *back-end*, precisava-se definir as rotas e operações de acordo com as solicitações da unidade. Por fim, no *front-end*, seria preciso não apenas utilizar os componentes comuns para montar o sistema, mas também implementar toda a parte visual, como especificado pela equipe de design.

A execução: O primeiro passo para a expansão foi adquirir os dados necessários para povoar o banco de dados, ou seja, os currículos Lattes dos docentes da FEA-RP. Desde o início da aproximação com a FEA-RP, um dos contatos da equipe era o chefe da STI do instituto, Kléber Benatti. Ele forneceu tanto os currículos quanto a relação dos

docentes e seus departamentos, sendo esta importante para que o *parser* soubesse separar corretamente os docentes no sistema.

Após a execução do *parser*, tanto o banco de dados completo em PostgreSQL quanto o de dados agregados em MongoDB estavam totalmente preenchidos, e portanto pôde-se iniciar o trabalho nas outras partes do sistema.

No *back-end*, foram poucas as ações necessárias. Visto que a FEA-RP havia solicitado um site igual ao da FAU, isso significava que todas as rotas e suas respectivas operações também seriam iguais. Dessa forma, bastou inserir no diretório da FEA-RP o mesmo arquivo de operações que estava sendo usado no diretório da FAU. Devido aos outros fatores já estarem definidos a partir da variável de ambiente (por exemplo, os nomes dos bancos de dados), não foi necessária nenhuma outra alteração.

Entretanto, é importante ressaltar que, no caso de um instituto desejar que uma operação seja realizada de outra forma, ou que uma nova operação seja inserida, bastaria alterar o arquivo com todas as operações no diretório desse instituto e inserir uma rota no sistema, no caso de uma nova operação. Desta forma, percebe-se que a criação do *back-end* para um novo instituto é feita de forma rápida, algo que sempre foi um dos maiores objetivos do projeto.

Já o *front-end* necessitou de um número maior de modificações, algo que já era esperado visto que essa é a parte do sistema que possui identidade visual e contato com o usuário e, portanto, deveria se adequar aos aspectos visuais da FEA-RP. Por conta de toda essa questão estilística, essas alterações só poderiam ser de fato implementadas após a equipe de design projetar todo o *layout* do sistema.

Entretanto, como se sabia que a FEA-RP desejava um sistema igual ao da FAU, e o *front-end* estava modularizado a partir de componentes comuns, pôde-se iniciar o desenvolvimento concomitantemente ao trabalho da equipe de design. Isso porque os componentes comuns poderiam ser inseridos da mesma forma, e conforme as especificações de design fossem recebidas, bastaria alterar os parâmetros necessários na criação dos componentes para adequá-los ao visual correto. Sendo assim, essa parametrização apresentou-se como algo muito vantajoso, pois permitiu que as duas equipes pudessem trabalhar simultaneamente, acelerando a construção do sistema.

Após a definição de todos os aspectos visuais do site, a equipe de desenvolvimento passou a modificar o sistema da FEA-RP alterando todos os elementos necessários. Neste momento também houve bastante comunicação entre as equipes, com a de design esclarecendo quaisquer dúvidas quanto ao *layout* do sistema, e a de desenvolvimento fazendo as implementações e oferecendo o parecer técnico quanto às modificações apresentadas.

Visto que um dos principais objetivos do projeto sempre foi a expansão rápida para outras unidades, avaliou-se o tempo gasto para implementação do sistema da FEA-RP, por parte da equipe de desenvolvimento. Primeiramente têm-se os bancos de dados, que com a utilização do *parser* podem ser povoados imediatamente, bastando inserir os currículos e a relação entre docentes e seus departamentos. Dessa forma, assim que a FEA-RP forneceu esses elementos, o banco de dados já pôde ser povoado e estava pronto para uso. O *back-end* também foi implementado rapidamente, visto que todas as operações necessárias eram as mesmas da FAU. Essas duas partes do sistema foram implementadas em uma única reunião

da equipe. Já para o *front-end*, a partir do momento em que o design foi apresentado, foram necessárias apenas três reuniões para que ele fosse totalmente implementado. A primeira delas consistiu apenas da equipe de desenvolvimento, que seguiu todas as especificações de design e modificou o sistema de acordo, enquanto a segunda contou também com a equipe de design, para esclarecimento de dúvidas e discussão de alguns detalhes. A última foi realizada entre a equipe completa do projeto, para verificar todo o sistema antes de agendar a entrega com a equipe da FEA-RP. Houve também comunicação assíncrona entre as partes para detalhes mais pontuais e pequenas discussões.

É importante notar que, embora o processo tenha sido muito rápido, uma parte disso deve-se à semelhança do sistema da FEA-RP em relação ao da FAU. Sendo assim, não houve a necessidade de se criar novos componentes e operações, o que claramente reduziu ainda mais o tempo de desenvolvimento. Mesmo assim, essa velocidade mostra que diversos fatores de generalização e modularização estão muito bem preparados, permitindo um desenvolvimento e expansão muito rápidos, atingindo assim um dos objetivos do projeto.

Esse processo de implantação para a FEA-RP mostrou que a arquitetura e estrutura do sistema, além de outras decisões de projeto, realmente permitiram que se criasse um sistema modularizado e expansível, que pode ser estendido rapidamente para outras unidades. Considerando o tempo gasto para implementação do sistema, pode-se supor que um sistema novo, que seja diferente dos dois já existentes, consiga ser desenvolvido em aproximadamente duas semanas, algo extremamente vantajoso. Isso porque a organização do sistema permite que sejam necessárias poucas modificações, sendo estas a adição e modificação de operações no *back-end*, a adição e modificação de componentes no *front-end* e então a construção do sistema usando estes componentes. Além disso, essas modificações podem ser feitas de forma eficaz por conta das tecnologias empregadas no sistema, com o Django permitindo a adição de rotas de forma prática e o React permitindo a criação rápida e bem estruturada de novos componentes.

Conclui-se então que um dos principais objetivos do projeto foi atingido, de forma que o CPqs Abertas poderá se expandir para diversos institutos em um curto período de tempo, aumentando cada vez mais a divulgação das produções acadêmicas para além das fronteiras da universidade.

3.17 Automatização da coleta de dados

O problema: No início do projeto, a partir do código legado do FAU Aberta, a atualização dos dados era feita manualmente, tanto no download dos currículos Lattes quanto na execução do *parser*. Embora a equipe de desenvolvimento tivesse julgado que isso não era ideal, decidiu-se por manter esse processo por dois motivos: a baixa frequência de atualização e a ausência de uma forma de obter os currículos automaticamente.

Devido à frequência mensal de atualização dos dados, realizá-la de forma manual era possível sem prejudicar o desenvolvimento. Além disso, para fazer a atualização automaticamente seria necessária uma maneira de obter os currículos Lattes também de forma automática. Entretanto, a ferramenta que era utilizada para isso, o scriptLattes, havia parado de funcionar, devido à questões de segurança da plataforma CNPq, como a utilização

de CAPTCHA. Buscando soluções para esse problema, descobriu-se que não havia mais suporte para a ferramenta. Sendo assim, sem uma forma de extrair esses currículos de maneira automática, a automatização da coleta dos dados não seria possível.

O professor Goldman buscou outras formas de obter esses currículos, mas sem sucesso. Contudo, após as conversas com a equipe de tecnologia da FEA-RP, descobriu-se que as unidades possuíam sistemas que constantemente coletavam os currículos dos docentes e os armazenavam. Com isso, seria possível automatizar a coleta de dados e executar o *parser* automaticamente, povoando o banco de dados sem interação humana.

A solução: Dada a possibilidade de coleta automática dos currículos Lattes, decidiu-se por realizar a automatização da coleta de dados, para permitir que as equipes de tecnologia dos diferentes institutos do projeto não precisassem se preocupar em atualizar o sistema de forma manual.

A execução: Um requisito inicial necessário para automatizar o *parser* seria garantir que os dados só fossem atualizados se não houvesse nenhum erro durante o processamento, para evitar dados incongruentes ou que apenas alguns docentes não estivessem atualizados. Sendo assim, a primeira tarefa foi implementar esse tipo de segurança.

A seguir, aproveitando que o tratamento de erros já estava sendo aprimorado, optou-se por melhorar a detecção e reportação deles. Dessa forma, mesmo com a atualização automática, os responsáveis pelo gerenciamento do sistema podem ser informados de eventuais erros que acontecerem no *parser*, permitindo sua correção e reexecução.

Por fim, realizou-se a integração com os sistemas que possuíam os currículos Lattes, para sua obtenção. Estes estavam armazenados em formato binário em um banco de dados relacional comum, e portanto foi preciso não só extrair os dados como também processá-los para obter os devidos currículos em formato XML.

Tendo estas funcionalidades implementadas, foi possível fazer com que o sistema extraísse esses currículos do banco de dados, convertesse-os para o formato XML e executasse o *parser* para atualizar o banco, informando sobre quaisquer erros encontrados no processo. Assim, facilita-se o trabalho das equipes de tecnologia dos institutos que integram o CPqs Abertas.

3.18 Refatorações e otimizações

É importante ressaltar que, durante o desenvolvimento do trabalho, diversas práticas de métodos ágeis foram empregadas, e portanto havia frequente comunicação entre as equipes e constantes revisões de código. Sendo assim, diversos momentos do desenvolvimento foram voltados para refatorações, para melhorar legibilidade e organização; otimizações de performance; e correção de *bugs*, tanto daqueles que já existiam no código legado quanto daqueles que foram introduzidos durante as atividades da equipe.

Muitas dessas tarefas foram pontuais, e por isso não foram registradas em detalhes durante as descrições apresentadas neste capítulo. Entretanto, foram importantes para me-

lhorar cada vez mais a organização do projeto, deixando o código mais legível, aprimorando sua estruturação e tornando sua manutenção mais fácil.

Além disso, essas melhorias permitem que as próximas equipes que trabalharem no projeto tenham menos dificuldade em entender o código existente e modificá-lo, realizando correções de erros, outras refatorações e até a expansão para os outros institutos.

Capítulo 4

Interações da equipe

Durante o projeto, a equipe de desenvolvimento não trabalhou isoladamente, comunicando-se tanto com a equipe de design quanto com os diferentes clientes, isto é, membros dos institutos participantes. A forma com que essas interações ocorreram e seus objetivos estão descritos neste capítulo.

4.1 Equipe de design

A equipe de desenvolvimento esteve em contato constante com a equipe de design, visto que esta era responsável pelos aspectos gráficos de todas as funcionalidades adicionadas aos sistemas e pela identidade visual dos novos sistemas desenvolvidos.

Sendo assim, a comunicação frequente era necessária para que a equipe de desenvolvimento soubesse quais os elementos visuais deveriam ser criados e pudesse informar sobre características que possivelmente não poderiam ser implementadas em curto prazo. Além disso, esse contato era importante para que eventuais dúvidas fossem clarificadas e para apresentar as modificações feitas para validação da equipe de design.

No início do projeto, isto é, durante as modificações feitas no FAU Aberta, o contato com a equipe de design foi pequeno, visto que o *layout* do site já havia sido desenhado e implementado. Desta forma, conversas entre as equipes aconteceram somente nos momentos de criação de outros elementos, como os mapas nacionais e o FAQ. A equipe de design também ficou encarregada de criar um manual para a FAU instruindo os docentes ao preenchimento correto do currículo Lattes na plataforma CNPq. Para isso, houve contato entre ela e a equipe de desenvolvimento para que estes, que conheciam o *parser* e os dados, pudessem informar sobre os campos em que havia maior quantidade de erros no preenchimento, para que a maneira correta fosse reforçada no manual.

Na implementação do FEA-RP Aberta, as equipes começaram trabalhando separadamente, com o time de desenvolvimento preparando o sistema a partir do FAU Aberta, e o time de design esquematizando seu *layout*. Com este finalizado, a equipe passou a configurar toda a parte gráfica do sistema, seguindo as especificações. Então, uma reunião foi feita entre as duas equipes, para que todos os elementos fossem verificados e ficasse garantido que o sistema estava como proposto pelos designers. Nesta reunião aproveitou-se

também para sanar dúvidas a respeito das características do *layout* e para exibir à equipe de design quaisquer atributos que não houvessem sido implementados por dificuldades computacionais, para discutir alternativas.

É importante ressaltar que adições e modificações de elementos visuais não foram realizadas apenas nas novas funcionalidades do FAU Aberta e na criação do FEA-RP Aberta. Constantemente novas especificações de *layout* eram apresentadas pela equipe de design, tanto por conta de mudanças solicitadas pelos clientes quanto por decisão da própria equipe. Assim, havia também contato frequente para a realização dessas modificações.

A experiência de trabalhar ao lado de uma equipe de design foi bastante interessante. O principal fator é a diferença de mentalidade das equipes em relação ao *layout* de sistemas e a experiência de usuário. Assim, observar a perspectiva dos designers em relação a esses aspectos trouxe ensinamentos para a equipe de desenvolvimento. Além disso, proporcionou desafios interessantes durante a programação de fato, visto que muitas vezes as solicitações apresentadas não eram simples ou imediatas na questão de implementação, e portanto havia esforço da equipe para atender a essas especificações pedidas.

4.2 Clientes

Durante o desenvolvimento do projeto, trabalhou-se com três grupos de clientes diferentes, sendo estes da FAU, da FEA-RP e do IME. A comunicação com cada um desses grupos deu-se de maneira distinta, mas visando o mesmo objetivo: o desenvolvimento do sistema para sua respectiva unidade.

No caso da FAU, primeiro instituto em que se trabalhou o projeto, o início das comunicações se deu de forma diferente em relação aos outros dois institutos, visto que o sistema da unidade já existia, e eram solicitadas apenas novas funcionalidades e modificações. Assim, por exemplo, não houve uma reunião inicial com diversos membros da FAU para definir interesses e estruturar o sistema, visto que esse processo havia acontecido com as equipes anteriores.

Outro fator interessante é que um dos membros da equipe total do projeto é o professor Artur Rozestraten, presidente da Comissão de Pesquisa da FAU. Sendo assim, no início do desenvolvimento ele pôde apresentar as solicitações da FAU diretamente, sem ser necessária uma reunião com os membros do instituto. Esse aspecto possibilitou um desenvolvimento mais rápido, pois durante as próprias reuniões da equipe podia-se tomar decisões e apresentar resultados. Além disso, a equipe de design era composta de membros da FAU, que também podiam fornecer suas opiniões e auxiliar nos processos referentes aos elementos do sistema de sua unidade.

Já após o término da implementação foi então realizada uma reunião com diversos docentes da FAU para apresentação do sistema finalizado, demonstrando todas as modificações em relação à versão anterior e explicitando novas funcionalidades. Devido a todo o contato com o professor Artur, nesta reunião o sistema já estava como pretendido pela equipe da FAU e, portanto, pôde-se passar para a fase de implantação do sistema.

Esta fase foi resolvida diretamente entre a equipe de desenvolvimento e a equipe de tecnologia da FAU. Instruções para o *deploy* do sistema foram definidas e passadas para a

equipe da FAU, permitindo que esta implantasse o sistema. Esse processo demandou certa comunicação entre as partes devido a alguns problemas surgidos durante a implantação. Sendo assim, esse contato foi muito importante para que a equipe pudesse organizar corretamente todas as instruções de *deploy*, a partir da experiência da equipe da FAU, permitindo que estas estivessem bem estruturadas para os próximos institutos.

Finalizado o processo da FAU, seguiu-se para a FEA-RP. Sendo um instituto novo no projeto, foi necessária uma primeira reunião geral com seus membros para apresentação dos CPqs Abertas, buscando entender se havia interesse da FEA-RP em participar do projeto. Eles se mostraram interessados de imediato e, por isso, nessa mesma reunião, já foram decididos os detalhes de como o sistema seria implementado.

A FEA-RP mostrou interesse em um sistema semelhante ao da FAU em termos de organização e estrutura, diferindo-se apenas em identidade visual, que seria decidida pelos designers a partir da identidade da FEA-RP e de seu site principal. Sendo assim, houve pouco contato com os membros da unidade, visto que não havia outras decisões a serem tomadas.

Durante o desenvolvimento do sistema da FEA-RP, toda a comunicação concentrou-se no chefe da equipe de tecnologia, Kléber Benatti. Desta forma, todas as solicitações da equipe em relação a informações da FEA-RP eram tratadas com ele, como identidade visual, currículos dos docentes e textos para preenchimento do site. Além disso, ele foi o responsável por apresentar à equipe de desenvolvimento a possibilidade de coleta dos currículos a partir do banco de dados fornecido pela STI, que permitiria a execução automática do *parser*. Sendo assim, também houve comunicação com ele para obtenção de acesso ao sistema com os currículos e entendimento do seu funcionamento.

Por fim, foi iniciado o processo de implementação do sistema para o IME, e, desta forma, também foi necessária comunicação com a unidade. De forma semelhante à FAU, um dos membros do projeto é o professor Alfredo Goldman, presidente da Comissão de Pesquisa do IME. Sendo assim, não foi preciso realizar a reunião inicial com membros do IME, de forma que o professor já apresentou como se desejava que fosse a estrutura e organização do sistema do IME, neste caso, também semelhante a da FAU, com exceção de um tipo de produção (artística).

O professor Goldman também foi a ponte de comunicação com o IME, fornecendo e buscando todas as informações necessárias para a implementação. Isto foi muito útil visto que, estando ele envolvido diretamente com o projeto, estas informações eram obtidas e apresentadas rapidamente.

Trabalhar com clientes reais foi uma experiência muito interessante, pois demonstrou de forma verdadeira o processo de desenvolvimento de um software para terceiros, comportando todos os elementos solicitados. Além disso, todas as pessoas com quem a equipe teve contato foram muito solícitas, algo que foi muito importante para que o desenvolvimento ocorresse de maneira fluida, sem intervalos de tempo em que se dependia desses terceiros para continuação do projeto. Assim como no caso da equipe de design, o contato com estes clientes proporcionou desafios e aprendizados interessantes, visto que era necessário atender ao máximo todas as solicitações deles referentes aos sistemas de seus institutos.

Capítulo 5

Experiências Pessoais

Este capítulo apresenta a opinião pessoal de cada autor deste trabalho, em relação ao contato com outras equipes e clientes e à realização de um trabalho de desenvolvimento em grupo.

5.1 João Gabriel Lembo

"Primeiramente, em relação à equipe de design, pode-se dizer que foi realmente muito interessante trabalhar no projeto de forma conjunta. A diferença de experiências e mentalidade das duas equipes foi muito importante para o projeto, pois agregou diferentes conhecimentos e perspectivas, permitindo que o trabalho ficasse mais completo e que uma maior quantidade de detalhes recebesse a devida atenção. Não apenas a visão da equipe de design ajudou a dar forma ao projeto e aos sistemas, como também trouxe ensinamentos que poderão ser usados em novos projetos e trabalhos, algo bastante valioso. Além disso, ao tomar o controle sobre os elementos visuais e gráficos do sistema, os designers permitiram que nós pudéssemos nos focar inteiramente ao desenvolvimento, à modularização do sistema e, consequentemente, à execução deste trabalho. Por fim, todos os membros da equipe de design foram muito solícitos, participativos e respeitosos, tornando toda a experiência de trabalhar com eles muito gratificante.

Já quanto aos membros de outros institutos participantes do projeto, a relação de trabalho foi diferente, mas também muito importante. Todas as pessoas com que tivemos contato estiveram sempre muito dispostas a nos ajudar com tudo o que fosse necessário, o que permitiu que pudéssemos realizar o trabalho sem grandes empecilhos. A comunicação assíncrona acabou atrapalhando em certos momentos, por conta do atraso gerado na comunicação, mas mesmo assim a proatividade de todos garantiu que o trabalho seguisse de forma fluida. Tomando como perspectiva esses membros como clientes do projeto, também tivemos uma experiência muito interessante. Pudemos ver na prática como é elaborar um projeto para um cliente real, seguindo suas necessidades e solicitações, buscando sempre atender a tudo o que foi pedido. Aproveitamos também a oportunidade para manter as práticas de métodos ágeis, como o desenvolvimento em pequenas etapas seguido de exposição para o cliente do que foi feito, para avaliação e definição de novos passos. Sendo assim, essa experiência também foi muito proveitosa.

Por fim, em relação a realizar o Trabalho de Conclusão de Curso em grupo, pode-se dizer que foi uma ótima experiência, e que trouxe muitos ensinamentos importantes. Criar o código em grupo traz diversos detalhes essencias, como a escrita de código com ótima legibilidade e manutenibilidade, a boa documentação dos passos realizados e do controle de versões, e a comunicação constante, de forma que os membros sempre se ajudaram para construir código de excelente qualidade. Todos esses aspectos são fundamentais para um bom programador, e tivemos a oportunidade de treiná-los e aperfeiçoá-los, algo que será muito proveitoso em futuros projetos em grupo, sejam estes acadêmicos ou no mercado de trabalho. Além disso, todos os membros do grupo sempre trabalharam com muita seriedade e dedicação, o que permitiu a elaboração de um trabalho de qualidade, que espera-se que tenha grande utilidade para a USP."

5.2 Leonardo Pereira

"O processo de interação com a equipe de design foi uma experiência enriquecedora, pois foi possível ver que realmente possuíam uma visão diferente da de desenvolvimento em termos de visual e aparência, observando detalhes que em geral passam despercebidos. Com a participação da equipe de design, foi possível focar totalmente na programação, o que proporcionou uma grande redução no tempo de escrita do código, visto que, normalmente, os detalhes de *layout* são verificados a todo momento, à medida que mais funcionalidades e partes do site são introduzidas. Contudo, parte dessa redução foi anulada por um aumento no tempo de espera pela confirmação de certos detalhes do *layout*, já que algumas partes do código teriam sua estrutura determinada por eles, dependendo do grau de complexidade.

Já a experiência com clientes reais, nesse caso os institutos, proporcionou uma noção dos tipos de demanda que são feitas em um projeto de software. Foi possível observar que determinadas demandas acabam sendo bem complexas, algumas vezes sendo necessárias soluções alternativas que sejam do agrado do cliente.

Também houve a interação com a equipe de TI da FAU e FEA-RP. Com relação à FAU, o contato foi feito por conta do *deploy* da aplicação em seus servidores. Foi necessário explicar todo o processo de implantação e também como alimentar a base de dados com novos dados para que fosse possível atualizar as informações dos docentes. Esse processo ajudou a enxergar problemas no processo de *deploy* como um todo, além de certas partes que poderiam ser facilitadas para torná-lo o menos incômodo e complexo possível. Quanto à FEA-RP, o contato se deu por conta da necessidade de informações sobre os docentes, como o departamento a qual cada um deles pertenciam, bem como seus currículos Lattes. Essa dependência causou um certo atraso no desenvolvimento, visto que maiores esclarecimentos sobre detalhes do instituto não eram obtidos de forma imediata.

Fazer este trabalho final em grupo também foi algo interessante, pois por ser um projeto de grande porte, precisamos planejar bem a forma como iríamos trabalhar, de forma que pudéssemos ter o maior rendimento possível, mas ao mesmo tempo evitar problemas em excesso no código devido a possíveis revisões demoradas. Optamos por fazer inicialmente sessões de *Mob Programming*, as quais puderam proporcionar uma boa experiência de trabalho em grupo, como por exemplo situações de impasse entre os membros do grupo

e quais decisões tomar quando elas ocorriam. Eventuais problemas criativos no código eram resolvidos rapidamente, devido ao fato de que havia três pessoas colaborando, e, consequentemente, menos erros passavam despercebidos."

5.3 Victor Lima

"Acredito que o primeiro grande desafio do projeto se deu justamente pela questão de continuar um projeto já existente, desenvolvendo sobre um código legado. Durante a maioria do curso, os projetos são desenvolvidos do início ao fim por nós mesmos, tanto individualmente quanto em grupo. Visto isso, desenvolver justamente o Trabalho de Conclusão de Curso sobre um projeto legado se mostrou uma experiência valiosa, induzindo um cuidado ao se olhar o código de outra pessoa, buscando entendê-lo e não apenas substituí-lo, além de preparar os desenvolvedores do grupo para o cenário mais comum do mercado de trabalho.

Outra experiência enriquecedora que se diferencia muito do que vemos no nosso dia a dia como programadores durante o progresso do curso foi o trabalho em conjunto com os designers da FAU. Por se tratarem de alunos de outro instituto, eles constroem uma visão acerca do projeto de um jeito muito peculiar se comparado ao nosso, levando em conta aspectos como o histórico e a 'personalidade' daquele instituto, por exemplo. Além disso, a atenção aos menores detalhes, que geralmente nos passavam despercebidos, permitiu um belo resultado final, criando um design geral que tem um visual muito característico e marcante para o projeto como um todo, mas que também se diferencia nos mínimos detalhes quando se compara instituto por instituto.

Além disso, ter os institutos como nossos clientes foi um grande prazer e um grande aprendizado também. Na maior parte da graduação, o que mais se assemelha a um 'cliente' são os nossos professores, salvo algumas exceções. Nessa dinâmica, geralmente temos um enunciado estático do que se deve fazer e temos nossas entregas avaliadas depois, uma dinâmica simples e fixa. Já no projeto, ocorreu exatamente o contrário: uma dinâmica viva, com pedidos constantes dos clientes, mudanças espontâneas nesses pedidos de acordo com a visão deles, entregas e a avaliação delas de acordo com o nível de satisfação e usabilidade que aquilo agregaria aos clientes.

Acredito que a soma dessas experiências mostra que todas as partes são importantes e necessárias para a elaboração do projeto. Pudemos perceber como a comunicação entre equipes e com os clientes é importante para realizar um projeto de forma a se tornar agradável para os desenvolvedores e satisfatório para os clientes.

Por fim, não posso esquecer da sinergia entre nós desenvolvedores, algo essencial para o desenvolvimento do projeto e também para o bom andamento do TCC como um todo. Finalmente fui convencido de que *Mob Programming* é algo que pode ser muito benéfico, visto as vantagens de se ter optado por esse método no início do projeto. Após isso, a química entre nós já estava bem estabelecida e pudemos dar os passos finais para essa etapa do projeto, deixando uma boa fundação para o futuro próximo. Hoje, entendo que não teria escolhido outras pessoas para realizar esse projeto."

Capítulo 6

Próximos passos e conclusão

O projeto CPqs Abertas tem como objetivo auxiliar todos os institutos da USP e, portanto, toda a equipe espera que continue sendo desenvolvido por muito tempo. Desta forma, existem diversos aspectos do projeto, no âmbito de desenvolvimento, que podem ser mais explorados no futuro para melhorar a qualidade dos sistemas. Alguns destes aspectos serão apresentados neste capítulo.

6.1 Migração do banco de dados

Uma possível mudança que pode ser realizada é a migração completa do banco de dados do sistema para o MongoDB. Durante a criação do banco de dados agregados para otimização das rotas, foram escolhidas apenas aquelas que envolviam algum tipo de contagem, mas verificou-se que também se poderia atuar sobre as operações das outras rotas. Dessa maneira, removeria-se a necessidade de dois sistemas de bancos de dados, reduzindo a complexidade do sistema e facilitando sua organização e estruturação.

Este processo não foi realizado por completo devido a outras demandas de maior prioridade durante o desenvolvimento do projeto, principalmente a implementação para um novo instituto (FEA-RP) e o início da criação para outra nova unidade (IME). Dada a importância dessas tarefas e o fato de que as rotas com maior tempo de espera já haviam sido otimizadas, optou-se por manter o sistema dessa forma, sem migrá-lo para MongoDB por completo. Avaliou-se também que a totalidade da migração teria dificuldade maior do que apenas o tratamento de algumas rotas, e portanto o tempo envolvido nessa tarefa seria grande, de forma que dedicar os esforços da equipe a essa atividade não seria vantajoso.

Entretanto, com maior tempo de desenvolvimento nos anos futuros do projeto e possíveis tempos de espera durante a implementação dos institutos, realizar tal migração seria possível, visto que as configurações e o uso inicial já foram iniciados. Assim, não só haveria ganho de desempenho, como também haveria redução na complexidade do entendimento e da manutenção dos sistemas. É importante ressaltar que antes de realizar esse procedimento seria necessário garantir que a mudança de sistema de bancos de dados realmente traria um ganho de performance, visto que talvez algumas operações tenham

melhor desempenho em PostgreSQL.

6.2 Unificação do banco de dados e *back-end*

Outra evolução no sistema que poderia ser realizada é a unificação completa do *back-end* e do banco de dados. Atualmente, dado que cada unidade executa o sistema dentro de sua própria rede, através da sua equipe de tecnologia, tem-se que cada instituto possui um sistema composto de um banco de dados, um *back-end* e um *front-end* próprios. Este último, devido a sua especificidade, não pode ser unificado, mas os outros dois serviços do sistema apresentam essa possibilidade.

O sistema de banco de dados agregados, em MongoDB, já é configurado de forma que seria possível utilizar o mesmo serviço para os sistemas de múltiplas unidades, visto que já é criado um banco de dados para cada unidade. Já o sistema do banco de dados total, em PostgreSQL, não está implementado dessa forma. Durante o *deploy* do sistema, um banco de dados específico para aquele instituto é criado, e portanto não pode haver múltiplos bancos em uma única instância. Entretanto, o PostgreSQL possui capacidade para esse tipo de funcionamento, e portanto a unificação é possível, de forma que uma única instância do sistema gerenciador possua um banco de dados para cada unidade. De outra maneira, a tarefa de migração completa para o MongoDB também aceleraria essa unificação, visto que esse sistema já está pronto para isso.

Já no caso do *back-end*, cada uma de suas instâncias está intrinsecamente ligada àquela unidade que a executou. No caso do banco de dados completo, o *back-end* recebe seu endereço através de uma variável de ambiente pré-configurada, de forma que ele se conecta diretamente ao banco daquela unidade. No caso dos dados agregados, por sua vez, ele utiliza o nome do instituto, através da variável de ambiente que informa a unidade para a qual o sistema está sendo executado, para localizar o banco de dados correto na instância do MongoDB.

Sendo assim, seria necessário remover essa conexão, fazendo-se com que o *back-end* conecte-se dinamicamente aos bancos de dados, a partir de informações sobre o instituto que está executando determinada operação. Com isso, as requisições feitas a ele poderiam incluir o instituto referente ao *front-end* que fez a requisição, e a partir dessa informação ele poderia se conectar ao banco de dados correto, realizar o processamento referente àquela unidade e retornar os dados formatados.

Essa unificação já havia sido planejada para o longo prazo, quando o projeto possuísse múltiplos institutos, antecipando uma eventual migração de responsabilidades das equipes de tecnologia específicas para a Superintendência de Tecnologia da Informação da USP (STI-USP). Isso porque, conforme o sistema crescesse e múltiplas unidades o integrassem, talvez fosse interessante para a USP unificar a responsabilidade dos sistemas, ao invés de deixar cada unidade mantendo o seu próprio, através de sua equipe de tecnologia. Sendo assim, um banco de dados e *back-end* unificados apresentariam grande vantagem, pois a STI-USP poderia executar apenas uma instância de cada um deles, ao invés de uma instância para cada instituto envolvido no projeto. Assim, haveria não só redução na complexidade da organização e da manutenção, como também diminuição dos recursos necessários para a execução do sistema.

6.3 Prosseguimento do IME Aberto

Assim, como o FEA-RP Aberta, o IME Aberto também é um sistema muito semelhante ao FAU Aberta, diferenciando-se apenas na questão da identidade visual. Sendo assim, iniciar o desenvolvimento para essa nova unidade foi uma tarefa simples, seguindo os mesmos passos da FEA-RP.

Desta forma, foram obtidos os currículos dos docentes do IME, para que o povoamento dos dados pudesse ser realizado. Além disso, as operações do *back-end* foram organizadas para o IME, algo que, assim como a FEA-RP, envolveu apenas a duplicação das mesmas operações utilizadas na FAU. É interessante ressaltar que essa duplicação é necessária, visto que, futuramente, algum dos institutos pode mudar a forma com que uma das operações é realizada, e por isso é preciso mantê-las separadas por instituto, mesmo que elas sejam iguais em um primeiro momento.

Já no *front-end* não foram feitas modificações, visto que os componentes comuns já fazem parte do sistema, e os específicos ainda não puderam ser criados, já que a equipe de design ainda estava trabalhando na identidade visual do sistema. Portanto, o próximo passo imediato para a futura equipe de desenvolvimento é a finalização do IME Aberto, criando o *front-end* do instituto a partir dos componentes comuns, advindos da modularização, e das especificações visuais apresentadas pela equipe de design.

6.4 Conclusão

O próximo passo mais importante para a continuidade do projeto é, obviamente, a expansão do sistema para novos institutos. Após o IME, já há alguns institutos mapeados para prosseguimento do projeto, como o Instituto de Física da USP (IF), a Faculdade de Direito de Ribeirão Preto (FDRP) e o Instituto de Arquitetura e Urbanismo da USP (IAU). Estas unidades demonstraram seu interesse após apresentação do CPqs Abertas em uma reunião da Pró-Reitoria de Pesquisa da USP (PRP), no dia 15 de dezembro de 2021. Além disso, espera-se que através de apresentações do projeto para a comunidade USP e com o apoio da PRP, outras unidades também se integrem ao CPqs Abertas. Embora ainda haja um longo período de tempo até a conclusão do projeto, visto o grande número de institutos na universidade, espera-se que o processo de inserir uma unidade seja bastante rápido e eficaz, dado o curto tempo necessário para a expansão do sistema para a FEA-RP e o início do desenvolvimento para o IME.

Através destes, verificou-se que a organização do sistema estruturada durante o ano de fato permite a extensão rápida para outras unidades, facilitando o trabalho das equipes de desenvolvimento. Com isso, haverá mais tempo disponível para a criação de novas funcionalidades e se permitirá que mais institutos possam divulgar suas produções para a comunidade em um menor tempo.

Já na eventualidade de uma unidade solicitar um sistema muito diferente dos já existentes, tem-se que será necessário um tempo maior para realizar a expansão, visto que isso exigiria criar novas visualizações e possivelmente funcionalidades. Contudo, conforme à adesão de novas unidades, o número de componentes do sistema, e consequentemente

suas funcionalidades e exibições disponíveis, será cada vez maior. Assim, aumenta-se a chance de que o site de uma nova unidade possa ser construído apenas com elementos presentes no sistema, acelerando o tempo de produção.

Desta forma, o CPqs Abertas poderá atingir seu objetivo primário em menor tempo, sendo este o de transmitir as produções acadêmicas da universidade para além de suas fronteiras, através de sistemas dinâmicos e visualmente agradáveis, criados de forma simples a partir de uma estrutura modularizada.

Apêndice A

Termos técnicos

Os seguintes termos são apresentados em inglês durante todo o texto, por serem usualmente utilizados desta forma no contexto de engenharia de software e/ou pela ausência de termo apropriado em língua portuguesa. Sendo assim, apresentam-se breves definições desses termos:

Back-end: Camada de um sistema responsável por acesso e manipulação de dados.

Commit: No contexto do Git, é uma operação que registra todas as mudanças realizadas entre dois estados consecutivos de um repositório.

Deploy: Preparação, instalação, implantação e disponibilização de um sistema em uma determinada máquina.

Framework: Estrutura de código genérica que tem seu comportamento definido e alterado por partes de código específicas escritas por seus usuários.

Front-end: Camada de um sistema responsável pela apresentação dos dados.

Log: Documento de registro de eventos e atividades de um sistema.

Mob Programming: Modalidade de programação em que um time de pessoas colabora na escrita de código simultaneamente, sendo uma pessoa responsável por escrevê-lo de fato enquanto as outras a auxiliam e se atentam a possíveis erros.

Model: No contexto deste trabalho, refere-se a uma classe Python que representa uma entidade do banco de dados.

Parser: Programa que converte os dados de um determinado formato para outro que seja manipulável por código.

Pipeline: Conjunto de processos automáticos encadeados para o teste, construção e/ou implantação de um programa ou sistema.

Proxy: Servidor intermediário entre o cliente e o servidor final de um determinado recurso, que pode apenas redirecionar ou modificar a requisição original.

Script: Programa ou sequência de instruções que são interpretados e executados por um outro programa.

Server-side: Refere-se ao lado do servidor no modelo cliente-servidor.

Setup: Processo de preparação e configuração de um sistema.

Apêndice B

Scripts

Programa B.1 *Script* de *setup* do sistema.

```
1   #!/usr/bin/env bash
2
3   # Set up the production environment. It is supposed to be run only once.
4
5   INSTITUTE_NAME=${INSTITUTE:-none}
6   if [[ $INSTITUTE_NAME == "none" ]]; then
7     echo "Instituto não especificado. Atribua um valor àvariável \$INSTITUTE."
8     exit
9   fi
10
11  DB_USER=${DB_NAME:-user}
12  DB_PASSWORD=${DB_PASSWORD:-password}
13  DB_NAME="${INSTITUTE_NAME}_db"
14
15  cd ./src/
16
17  echo ""
18  echo "########################################################################"
19  echo "# #"
20  echo "# Script para configuração da aplicação... Deve ser executado apenas
        uma vez #"
21  echo "# #"
22  echo "########################################################################"
23
24  echo ""
25  echo "Montando imagens..."
26  docker-compose -f docker-compose.prod.yml build
27
28  echo ""
29  echo "Inicializando containers..."
30  docker-compose -f docker-compose.prod.yml up -d
31
32  sleep 60
33
```

cont ⟶

⟶ *cont*

```
34    echo ""
35    echo "Rodando migrations..."
36    docker-compose -f docker-compose.prod.yml exec server bash -c "python manage.
          py migrate"
37
38    echo ""
39    echo "Aplicação está operante. Populando banco de dados..."
40    docker-compose -f docker-compose.prod.yml exec server bash -c "python util/
          populate/code/parser.py -v"
41
42    echo ""
43    echo "Finalizado"
```

Programa B.2 *Script* para medição de tempo de processamento das rotas.

```
1     #!/bin/bash
2
3     ROUTES=... #Rotas ocultadas para facilitar a legibilidade
4
5     for route in ${ROUTES[*]}; do
6       measures=""
7       average_time=0
8       for i in $(seq 1 100); do
9         start_time=$(date +%s%N)
10        curl --silent --output /dev/null $route
11        end_time=$(date +%s%N)
12        ellapsed=$(($($end_time - $start_time)) / 1000000))
13        measures+=",$ellapsed"
14        average_time=$($average_time + $ellapsed))
15        sleep 2
16      done
17      echo -e "\$route: \$measures\\n"
18      echo -e "\$route average: \$((\$average_time / 100))"
19    done
```

Referências

[Escobar 2019] Herton Escobar. *Jovens defendem a ciência, mas desconhecem produção científica do País*. 2019. URL: https://jornal.usp.br/universidade/politicas-cientificas/jovens-defendem-a-ciencia-mas-desconhecem-producao-cientifica-do-pais/ (acesso em 23/10/2021) (citado na pg. 1).

[Martin 2017] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Robert C. Martin Series. Boston, MA: Prentice Hall, 2017. ISBN: 978-0-13-449416-6. URL: https://www.safaribooksonline.com/library/view/clean-architecture-a/9780134494272/ (citado na pg. 3).