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

Refatoração de um sistema de análise de dados de produtividade científica

Paulo Henrique Silva Araujo

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Orientador: Marcelo da Silva Reis

São Paulo
29 de janeiro de 2021

Resumo

A produtividade científica é a medida do desempenho de um pesquisador, ao longo de um determinado período de tempo, em atividades intrínsecas a essa carreira, tais como publicação de artigos, captação de recursos, formação de alunos, entre outras. A avaliação da produtividade científica utiliza ferramentas cientométricas e é feita por instituições de pesquisa para decidir, entre outras coisas, a alocação de recursos e a progressão de carreira de pesquisadores. No Instituto Butantan, importante instituição de pesquisa biomédica do Estado de São Paulo, a avaliação de produtividade científica é feita anualmente, com cada pesquisador da instituição preenchendo um formulário que cobre todas as atividades científicas exercidas ao longo do ano. Esses formulários posteriormente são unificados e consolidados em um relatório final pela diretoria científica do instituto. Por muitos anos, esse formulário foi preenchido e consolidado de forma manual, utilizando planilhas Excel. Mais recentemente, foi desenhado um sistema de informação para realizar essa tarefa. Chamado de Scientiometer, o sistema foi implementado principalmente em Rails e conta com uma amigável interface web para facilitar o preenchimento e a consolidação do relatório. Todavia, antes desse sistema ser colocado em produção, foram detectadas inconsistências entre a especificação e a implementação do sistema. Tais inconsistências, se não corrigidas, acarretariam em comprometimento do uso do sistema por parte dos pesquisadores. Dessa forma, neste projeto foi proposta a refatoração do sistema Scientiometer, de forma a corrigir as inconsitências de implementação mencionadas acima. Para esse fim, foi feito o uso de boas práticas de programação e de refatoração, além da promoção de melhorias em pontos específicos do sistema. Nesta monografia é detalhado todo o processo de refatoração do sistema e também são mostrados alguns exemplos dessas correções e melhorias. Espera-se que, com este trabalho, o Scientiometer possa ser colocado em produção, para assim atender à demanda por uma maior eficiência na coleta e análise de dados de produtividade científica.

Sumário

Capítulo 1

Introdução

O empreendimento científico envolve diversas atividades acadêmicas, tais como publicações de artigos e livros em revistas especializadas, orientações de alunos, ministração de aulas e cursos, captação de recursos, prestação de serviços, entre outros. Combinações quantitativas desses elementos podem definir índices para qualificar a produtividade científica de um pesquisador. Universidades e outros institutos de pesquisa fazem uso desse tipo de avaliação para definição de alocação de recursos e também para definição de progressão de carreira de seus pesquisadores. Nesse sentido, o Instituto Butantan (IB) possui um relatório anual, que é preenchido individualmente pelos seus pesquisadores. Esse relatório é coletado, consolidado e organizado pelo Centro de Desenvolvimento Científico (CDC) do IB, sendo utilizado inclusive como forma de prestação de contas da atividade científica do Instituto junto à sociedade paulista, por meio do envio do relatório para a Secretaria de Saúde do Estado de São Paulo.

O relatório é estruturado em sete passos: primeiro, são coletados dados sobre o pesquisador; segundo, de funcionários e alunos; terceiro, de artigos, livros e congressos; quarto, de teses, disciplinas e cursos; quinto, de recursos captados; sexto, de atividades administrativas, culturais de inovação ou prestação de serviços e sétimo, de observações gerais. Primordialmente, o relatório era preenchido de maneira mista em formatos doc (Word) e xls (Excel). Num segundo momento, criou-se um modelo exclusivamente em xls; isso facilitou seu preenchimento, porém é altamente suscetível à quebra dos padrões ou normalizações pretendidas pelos criadores do arquivo original. Em ambos os momentos, a análise dos dados também era impactada negativamente, uma vez que existem muitos campos de texto livre (difícil agregação) ou adulterados (pela pessoa que preenche o relatório), com valores não existentes nos conjuntos originais (erros de agregação). Outro ponto problemático é o de consolidação dos relatórios preenchidos, ou seja, a agregação dos dados preenchidos por todos os pesquisadores, pois são muitos arquivos a serem administrados.

No ano passado, visando sanar os pontos mencionados acima, foi iniciado o desenvolvimento de um sistema web para coleta, aprovação e análise dos relatórios de produtividade científica dos pesquisadores do Instituto Butantan. O sistema, batizado de Scientiometer (Scholl, 2019), possui três estruturas: um banco de dados relacional e persistente, PostgreSQL (Momjian, 2001), para armazenamento dos relatórios, ou seja, os dados das

entidades abstratas presentes neles (alunos, pesquisas, congressos etc), além dos metadados e outros necessários para o funcionamento do fluxo de *login* (autenticação, autorização, sessão e segurança) e associação dos dados aos pesquisadores (pesquisadores, contas, funcionários, laboratórios etc); um *back-end* escrito em Ruby, utilizando o *framework* Rails (Ruby on Rails) (Taylor, 2010), utilizado para *CRUD* (criação, extração, atualização e deleção, em inglês) de todos os dados do sistema, também para comunicação com o *front-end* (explicado a seguir), para recebimento e envio dos dados, em formato *json*, via APIs HTTP, e, finalmente, o *front-end* (figura 1.1), desenvolvido em JavaScript, com o *framework* Vue.js (Filipova, 2016), uma GUI (interface gráfica de usuário, em inglês) *online*, para os pesquisadores acessarem e preencherem o formulário e os responsáveis realizarem aprovação.

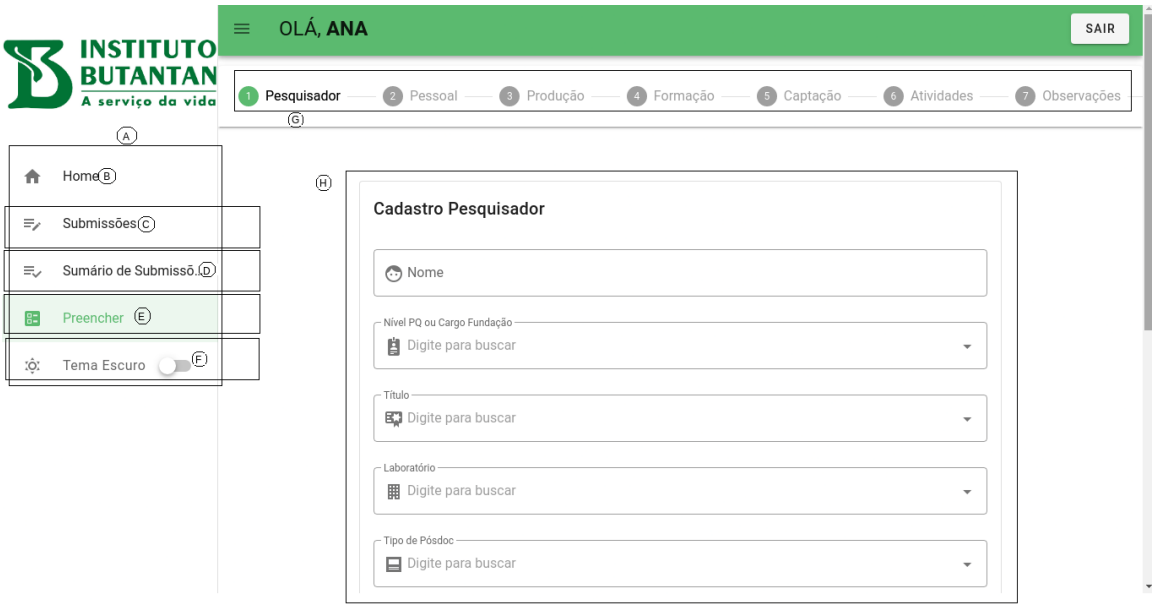


Figura 1.1: Screenshot *da interface de Scientiometer, no qual é mostrado o preenchimento do passo 1 na elaboração de um relatório de produtividade (cadastro de pesquisadores). (A) Barra lateral; (B) entrada; (C) relatórios submetidos aguardando aprovação; (D) quantidade de relatórios submetidos e aprovados, por laboratório; (E) preenchimento do relatório (tela atual); (F) alterar esquema de cores; (G) Passo-a-passo do relatório e (H) local para o preenchimento.*

A parte do banco de dados do Scientiometer foi utilizada ainda no ano passado em produção, para análises cientométricas, por meio de consultas SQL, após preenchimento *offline* com os dados do relatórios produzidos em arquivo xls. Portanto, as estruturas *back* e *front* desse sistema não tiveram utilização pelos usuários finais. Além disso, essas estruturas contavam com poucos testes unitários e também não passaram por testes de integração, visando a sua aprovação para colocação em modo de produção. A estes fatos soma-se a constatação, durante o prosseguimento da construção do projeto retomada neste ano, da necessidade de refatoração do sistema, seja por motivos de readequação (e.g. simplificar relacionamentos entre entidades), de melhorias (e.g. possibilitar aos pesquisadores a capacidade de submeter, novamente, à aprovação, relatórios reprovados) ou de correções (e.g. certas estruturas não eram persistidas).

1.1 Objetivos

Esse trabalho tem como objetivo geral a refatoração do sistema Scientiometer, para que o mesmo possa ficar pronto para ser colocado em produção no Instituto Butantan e mesmo em outras instituições de pesquisa.

Também são propostas, também por meio de técnicas de refatoração, as seguintes metas específicas:

- a correção do fluxo completo de preenchimento e aprovação do formulário de produtividade científica;
- permitir que o sistema possa ser evoluído futuramente de forma objetiva (realizar o esperado), segura (garantir a integridade das funcionalidades) e escalável (aberto às novas demandas).

1.2 Organização da monografia

O restante desta monografia está organizada da seguinte maneira:

- **Capítulo 2:** recapitulação de conceitos fundamentais para o acompanhamento do restante desta monografia, tratando brevemente de conceitos cientométricos e também de metodologias de refatoração de código;
- **Capítulo 3:** apresentação da metodologia do trabalho, mais especificamente o fluxo de processos utilizado ao longo da refatoração do Scientiometer;
- **Capítulo 4:** exemplificação de resultados obtidos com a refatoração, colocando ênfase nas principais inconsistências que foram sanadas;
- **Capítulo 5:** recapitulação do conteúdo desta monografia, além de apontamentos de possíveis melhoramentos futuros nesse sistema.

Capítulo 2

Conceitos fundamentais

Nesse capítulo são apresentados conceitos fundamentais para facilitar a leitura do restante deste trabalho. No início, são introduzidos conceitos essenciais de cientometria. Na sequência, são abordadas técnicas de refatoração de código.

2.1 Cientometria

Cientometria é o campo de estudos que lida com a medição e quantificação da produção científica (Silva e Bianchi, 2001). Como todo campo, ela é permeada de jargões que abrangem diversos aspectos da vida acadêmica. A seguir serão explicados alguns deles, pois os mesmos são relevantes na elaboração de um relatório científico. Vários desses conceitos serão abordados aqui de forma resumida porque os mesmos já foram trabalhados em Scholl, 2019; dessa forma, esse trabalho é recomendado para maior aprofundamento nesses conceitos.

2.1.1 Paper

Paper é um instrumento utilizado por cientistas para compartilhar o seu trabalho com outros cientistas ou revisar a pesquisa conduzida por outrém. Dessa forma, são parte crítica na evolução do conhecimento e necessários à avaliação de produtividade científica (Scitable, 2020).

2.1.2 Citação

Citação é o reconhecimento, por parte de um pesquisador, de que outro *paper* contribuiu para seu trabalho (Smith, 1981). Existem estudos explorando a análise de citações (Gupta, 1979); um trabalho citado um grande número de vezes é evidência de que o conhecimento gerado é altamente relevante.

2.1.3 Índice H

Índice H é um número para caracterizar a produção científica de um pesquisador. Um cientista tem índice *h*, se *h* de seus artigos tiverem pelo menos *h* citações cada e os outros artigos tiverem *h* ou menos citações cada. (Bornmann e Daniel, 2005)

2.1.4 Fator de Impacto

Fator de impacto de um periódico é uma medida quantitativa da relevância do mesmo. É baseado em 2 elementos: o numerador, que é o número de citações no ano atual para quaisquer itens publicados em um periódico nos 2 anos anteriores, e o denominador, que é o número de artigos substantivos (itens de origem) publicado nos mesmos 2 anos. É um método simples para comparar periódicos, independentemente de seu tamanho (Garfield, 1999). Normalmente um pesquisador procura publicar em periódicos de alto impacto, pois isso coloca seu trabalho em maior evidência do que publicando-o em um periódico de impacto menor.

2.1.5 Qualis

Qualis é um sistema usado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), para classificar a produção científica dos programas de pós-graduação no que se refere aos artigos publicados em periódicos científicos. Qualis afere a qualidade dos artigos e de outros tipos de produção, a partir da análise de qualidade dos veículos de divulgação, ou seja, periódicos científicos. Na Classificação de 2017-2020, os veículos foram classificados nos seguintes estratos: A1, mais elevado; A2; A3; A4; B1; B2; B3; B4; C - peso zero (CAPES, 2020). Pesquisadores brasileiros, para dar maior visibilidade a suas publicações, procuram sempre periódicos com estratos Qualis mais elevados.

2.1.6 Bolsa de Produtividade do CNPq

O Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) define suas bolsas de produtividade da seguinte maneira:

"O CNPq concede bolsas para a formação de recursos humanos no campo da pesquisa científica e tecnológica, em universidades, institutos de pesquisa, centros tecnológicos e de formação profissional, tanto no Brasil como no exterior.

Além de promover a formação de recursos humanos em áreas estratégicas para o desenvolvimento nacional, o CNPq aporta recursos financeiros para a implementação de projetos, programas e redes de Pesquisa e Desenvolvimento, diretamente ou em parceria com os Estados da Federação." (CNPq, 2020).

2.1.7 Nível PQ

Nível PQ é o nome comumente utilizado para se referir à denominação da divisão de cargos da carreira de pesquisador estatutário do Estado de São Paulo: I; II; III; IV; V e VI. Levam em conta: capacitação científico-tecnológica; desempenho de atividades específicas de investigação científica ou tecnológica, em nível de coordenação, orientação

e execução, e grau de complexidade e responsabilidade decorrentes do exercício das atribuições (Martins, 2020).

2.2 Refatoração

Refatoração é a alteração das estruturas internas do software sem modificar seu comportamento. São utilizadas técnicas para construir mudanças ao longo do código-fonte (Fowler, 2018). Esta seção contém pequenos exemplos (em Ruby on Rails) de aplicações de diversas técnicas de refatoração; nos capítulos posteriores, alguns desses exemplos serão referenciados durante a apresentação do código do refatoramento real do sistema Scientiometer.

2.2.1 Composição de funções

Um programa pode ser entendido como uma sequência de funções, portanto refatorar essas entidades deve ser visto como prioritário e muito do tempo investido será nessa melhoria. Os exemplos a seguir auxiliam nessa organização e possibilitam modificações futuras.

Múltiplas responsabilidades - Extração de função

Por vezes, são acrescentadas funcionalidades a um mesmo nome (i.e. função). Isso pode ser visto quando o nome não reflete tudo o que está ocorrendo no seu corpo (e.g. uma função chamada *preço*, além de calcular o preço, possui a lógica para calcular o lucro a ser embutido no valor final, tornando o nome *preço* apenas parcialmente verdadeiro). Nesses casos, deve-se identificar quais lógicas precisam estar fora da atribuição local e delega-las a outras funções, posteriormente substituir o código inicial por suas chamadas.

```
1   def preco(valor, tipo_de_lucro)
2      if tipo_de_lucro == 'simples'
3         return valor*1.1
4      elsif tipo_de_lucro == 'complexo'
5         return valor*1.2
6      end
7      valor*1.3
8   end
```

```
1   def preco(valor, tipo_de_lucro)
2      acrescimo_tipo_de_lucro = acrescimo(tipo_de_lucro)
3      return valor*acrescimo_tipo_de_lucro
4   end
5
6   def acrescimo(tipo_de_lucro)
7      if tipo_de_lucro == 'simples'
8         return 1.1
9      elsif tipo_de_lucro == 'complexo'
10        return 1.2
11     end
12     1.3
13  end
```

Atribuição a parâmetro - Variável local

(Re)Atribuir um valor a um parâmetro no corpo da função pode levar a um efeito colateral, dificultando o raciocínio sobre a lógica do programa e fazendo com que o chamador (= *caller*) tenha um resultado inesperado ao reutilizar o parâmetro em outros locais. Utilizar uma variável local com o argumento é um passo na direção de funções puras, metodologia que torna o raciocínio e testes de programa mais fáceis. Mesmo que a linguagem permita passagem por valor, evitar essa reatribuição condicionará o programador à boa prática.

```
1    def preenche_tabela(nome, numero, cidade)
2       if cidade == 'Osasco'
3          cidade = 'Osasco-SP'
4       // ...
5    end
```

```
1    def preenche_tabela(nome, numero, cidade)
2       local_cidade = cidade
3       if local_cidade == 'Osasco'
4          local_cidade = 'Osasco-SP'
5       // ...
6    end
```

Reatribuição de variável local - Múltiplas variáveis locais

Reatribuir uma variável local dificulta o entendimento, pois seu nome não será mnemônico; em partes diversas, terá significados diversos. Nesse caso, criam-se variáveis intermediárias onde cada uma possua uma interpretação exclusiva dentro daquele escopo.

```
1    valor = largura * profundidade
2    ...
3    valor = valor * altura
4    ...
```

```
1    base = largura * profundidade
2    ...
3    volume = base * altura
4    ...
```

2.2.2 Organização de dados

A escolha de um nome para um valor é essencial para a organização do código, pois ele explicita sua intenção, aumentando a eficiência no entendimento do programa, e abre portas à uma fácil refatoração, pois , com um nome associado, basta movê-lo ou modificar seu valor em apenas um local do código.

Número mágico - Constante

Com exceções de variáveis de iteração que possuem número fixo de passos (e.g. (i = 0; i < MAX ; i++)), um número possui um significado que pode não ser explícito ao leitor do

código. Isso pode ocorrer com constantes físicas (e.g. g = 9,8), medidas de aproximação (e.g. *épsilon* para definir números aceitáveis para arredondamentos) e outros casos.
É conveniente criar uma variável para guardar esse valor. Isso facilitará a compreensão e seu futuro refatoramento, se necessário.

```
1   def raiz_quadrada_newton(x)
2     estimativa = 1.0
3     while true
4         estimativa = (estimativa + x / estimativa) / 2
5         diferenca = (x - estimativa ** 2).abs
6         if diferenca <= 0.000001
7            break
8         end
9     end
10    estimativa
11  end
```

```
1   def raiz_quadrada_newton(x)
2     epsilon = 0.000001
3     estimativa = 1.0
4     while true
5         estimativa = (estimativa + x / estimativa) / 2
6         diferenca = (x - estimativa ** 2).abs
7         if diferenca <= epsilon
8            break
9         end
10    end
11    estimativa
12  end
```

Nome não mnemônico - Renomeação

Uma variável ou função cujo nome não explica o seu significado deve ser renomeada, para melhorar a legibilidade do leitor. Caso seja utilizada por outros clientes que o reponsável pela refatoração não tenha acesso, ao invés de renomear, cria-se outra função com o novo nome, porém a mesma funcionalidade e se marca a antiga como *deprecated*. Isso impedirá a quebra do contrato, ou seja, a interface que foi garantida para outros clientes deve ser mantida funcionalmente e só poderá ser removida caso seja explicitado com antecedência.

```
1   def calc(a):
2      2 * 3,14 * a
3   end
```

```
1   def circunferencia(raio):
2      2 * 3,14 * raio
3   end
```

2.2.3 Mover funcionalidades

As funcionalidades de um código, por vezes, precisam ser transportadas para outros locais. A seguir, são apresentadas formas seguras e recomendadas de realizar essa prática.

Múltiplas abstrações - Extração de classe

Uma classe deve ser a abstração de uma única entidade. Quando uma classe possui um conjunto de características que pode ser atribuída à outra abstração ou ainda é repetido em outras, ele deverá ser extraído para uma nova classe (= tipo).

```
class Pessoa
   attr_accessor :nome
   attr_accessor :rua
   attr_accessor :numero
   attr_accessor :cidade

   def initialize(nome, rua, numero, cidade)
      @nome = nome
      @rua = rua
      @numero = numero
      @cidade = cidade
   end
end
```

```
class Pessoa
   attr_accessor :nome
   attr_accessor :endereco

   def initialize(nome, endereco)
      @nome = nome
      @endereco = endereco
   end
end

class Endereco
   attr_accessor :rua
   attr_accessor :numero
   attr_accessor :cidade

   def initialize(rua, numero, cidade)
      @rua = rua
      @numero = numero
      @cidade = cidade
   end
end
```

Corrente de chamadas - Ocultar delegado

Um cliente de função não deve ser responsável por conhecer todo o fluxo dentro das estruturas para requisitar um valor. Esse anti-padrão (= *anti-pattern*) pode ser comumente encontrado em linguagens orientadas a objetos na forma *getA.getB.getC....*

Deve ser feita uma interface onde o cliente possa requisitar o valor diretamente, para evitar que alterações nas estruturas demandem alterações nas chamadas dos clientes.

```
class Departamento
   attr_accessor :gerente
   def initialize(gerente)
      @gerente = gerente
   end
end

class Funcionario
   attr_accessor :departamento
   def initialize(departamento)
      @departamento = departamento
   end
end

departamento = Departamento.new('Alice')
funcionario = Funcionario.new(departamento)

funcionario.departamento.gerente // Alice
```

```
class Departamento
   attr_accessor :gerente
   def initialize(gerente)
      @gerente = gerente
   end
end

class Funcionario
   attr_accessor :departamento
   def meu_gerente
      @departamento.gerente
   end
   def initialize(departamento)
      @departamento = departamento
   end
end

departamento = Departamento.new('Bob')
funcionario = Funcionario.new(departamento)

funcionario.meu_gerente // Bob
```

2.2.4 Condicionais

Uma das formas mais comuns de controle do fluxo dentro de um programa é a utilização de condicionais. Por esse motivo, eles tendem a se tornar mais complexos, menos abstratos e, consequentemente, menos inteligíveis. São apresentadas técnicas de simplificação de condicionais.

Múltiplos condicionais para o mesmo resultado - Consolidação de condicional

O controle de fluxo de um programa requer clareza ao leitor. Se há diversas condições finalizando no mesmo resultado, não fica evitente quais são as saídas do programa. Consolidar condições num mesmo nome pode facilitar a compreensão.

```
1    if pessoa.idade >= 18
2       true
3    elsif veiculo.novo
4       true
5    ...
```

```
1    if valida_habilitacao(pessoa, veiculo)
2       true
3    ...
4
5    def valida_habilitacao(pessoa, veiculo)
6       // implementação
7    end
```

Condicional - Polimorfismo

Costumeiramente, são introduzidas lógicas dentro de condicionais, ou seja, o código de condicional irá avaliar o dado para depois realizar determinada ação. Alinhada à orientação a objetos, é melhor dizer ao objeto o que ele deve fazer, ao invés de perguntá-lo qual é o seu estado atual (e.g., se ele está armazenando um número inteiro ou um string). Isso manterá a separação de atribuições; cada objeto deve ser responsável por conter suas lógicas e efetuar suas próprias ações.

```
1    class Carro
2       attr_accessor :tipo
3
4       def velocidade
5          if @tipo == 'lento'
6             10
7          elsif @tipo == 'medio'
8             20
9          else
10            30
11         end
12      end
13
14      def initialize(tipo)
15         @tipo = tipo
16      end
17   end
```

```
1    module Carro
2       def velocidade
3          fail NotImplementedError
4       end
```

```
5    end
6
7    class Lento
8       include Carro
9
10      def tipo
11         'lento'
12      end
13
14      def velocidade
15         10
16      end
17   end
18
19   class Medio
20      include Carro
21
22      def tipo
23         'medio'
24      end
25
26      def velocidade
27         20
28      end
29   end
```

Condicional complexo - Extração de função

À medida em que uma condição se torna maior e mais complexa, menos imediata ou significativa será a assimilação do leitor. Assim, deve-se transcrevê-la sob novo nome.

```
1    if veiculo.fabricacao > 1980 && nome == 'XPTO' || veiculo.preco >= 5000
2       'valido'
3    elsif
4       ...
```

```
1    if valida(veiculo, nome)
2       'valido'
3    elsif
4    ...
5
6    def valida(veiculo, nome)
7       // implementação
```

Capítulo 3

Metodologia

No início da proposta de prosseguimento do desenvolvimento do sistema Scientiometer, foi feito um estudo abrangente do código-fonte e da documentação do que foi implementado até então. Isso foi feito tanto por meio de engenharia reversa do código-fonte quanto com reuniões com o desenvolvedor original do sistema. Durante esse período de estudos, constataram-se inconsistências entre a especificação do projeto, apresentada em Scholl, 2019, e o código-fonte do sistema (figura 3.1), além da necessidade de melhorias diversas (como a visualização de relatórios já preenchidos; ver figura 3.2).

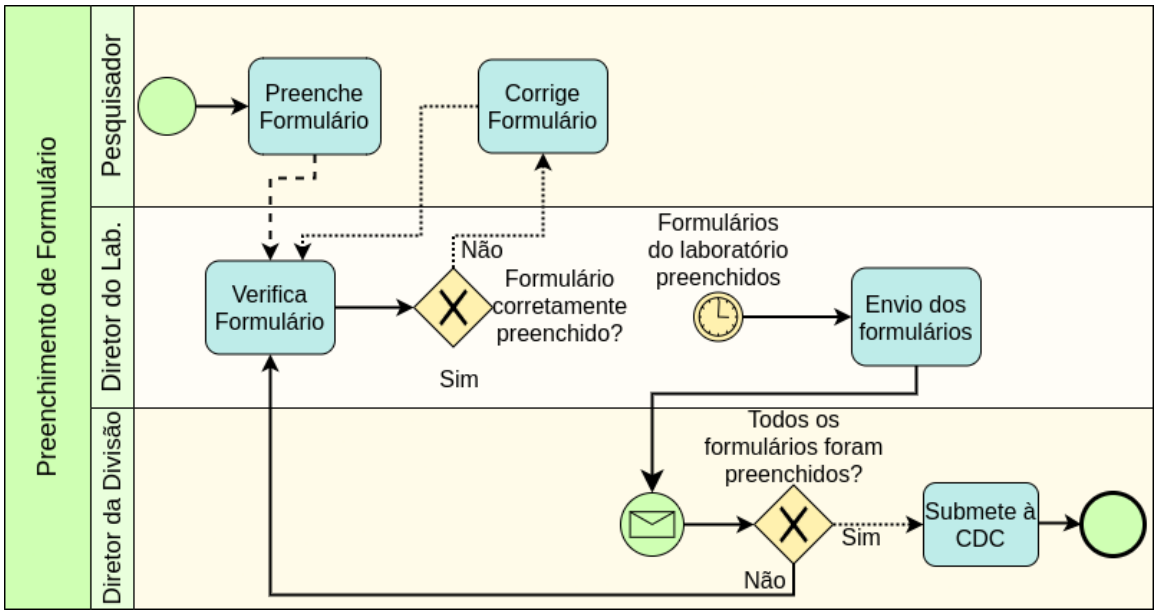


Figura 3.1: *Modelo e Notação de Processo de Negócio (BPMN, em inglês) que ilustra as regras de negócio do fluxo de preenchimento, validação e entrega dos formulários de produção científica do Instituto Butantan. A imagem foi retirada de Scholl, 2019 e modificada para indicar quais seções do fluxo de trabalho foram implementadas anteriormente no sistema Scientiometer: linha pontilhada indica ação não implementada; linha tracejada, parcialmente implementada e linha cheia, implemetada.*

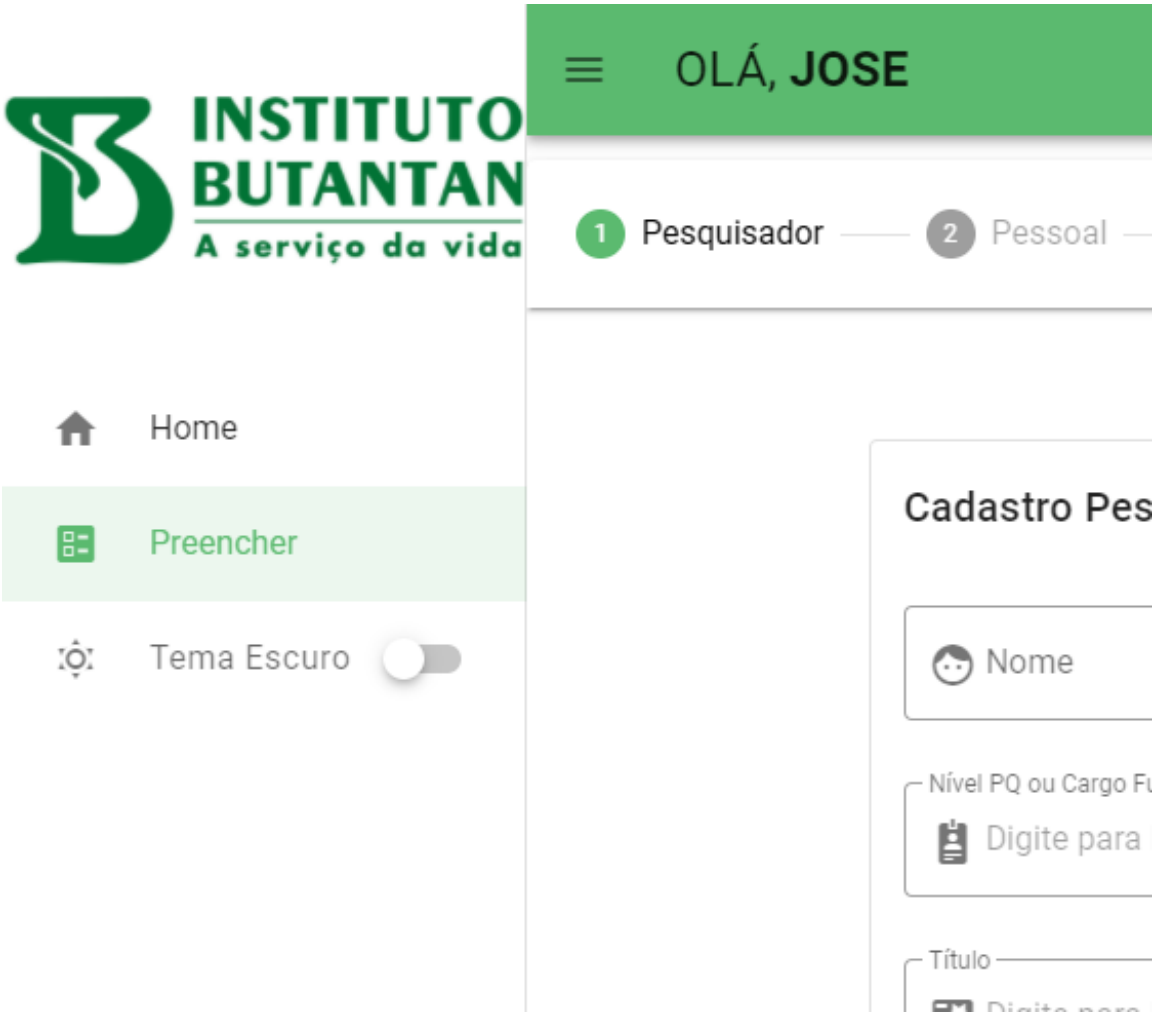


Figura 3.2: *Barra lateral de tarefas do sistema Scientiometer, antes da refatoração. Um pesquisador não poderia visualizar um relatório de produtividade preenchido anteriormente, havia apenas a opção de preenchimento.*

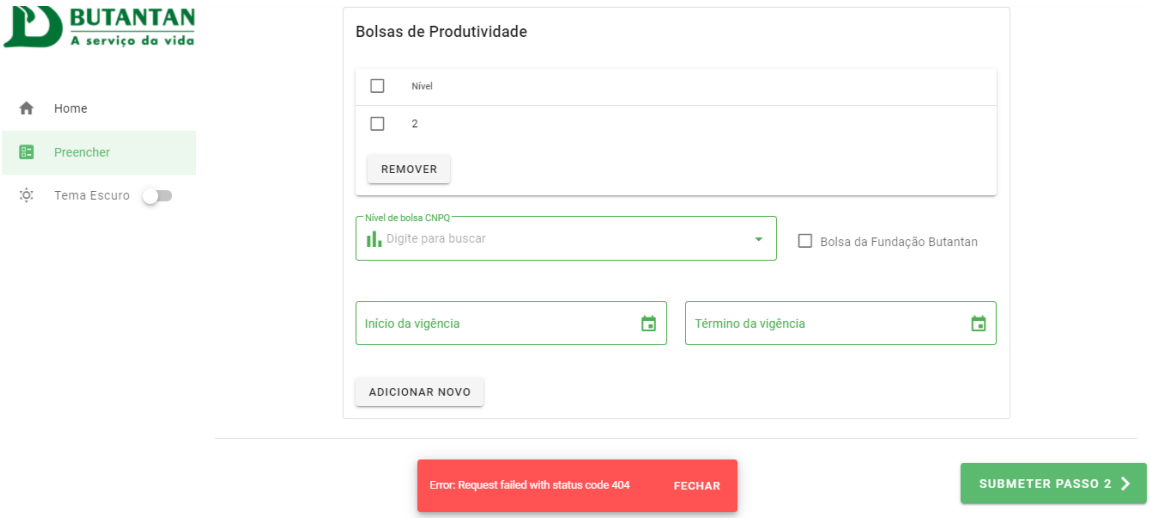


Figura 3.3: *Tela de preenchimento do passo 2 do relatório de produtividade no sistema Scientiometer, antes da refatoração. Não era possível preencher todo o relatório, pois havia erros inesperados, como o exemplo dessa figura, em que não era possível inserir uma bolsa de produtividade. Parte da solução desse erro é apresentada na figura 3.5.*

Para correção das inconsistências de implementação em relação à especificação do Scientiometer (como exemplo apresentado na figura 3.3), foi necessário levar a cabo uma abrangente refatoração. Para essa finalidade, necessitou-se, antes de tudo, preparar o sistema para essa empreitada, uma vez que não havia testes unitários e a documentação do código-fonte não estava adequada para modificações posteriores do sistema por parte de terceiros.

3.1 Preparação do sistema para refatoração

3.1.1 Repositório

Para administrar o desenvolvimento do sistema Scientiometer, utilizou-se o GitHub (Tom Preston-Werner, 2020); uma plataforma online e colaborativa para manter os códigos-fontes de projetos e seus históricos de alterações. O GitHub implementa o Git (Torvalds, 2020), um software de versionamento e rastreamento de alterações em arquivos.

3.1.2 Deploy em ambiente de testes

Para manter o sistema disponível na *web*, durante a fase de testes, manteve-se o *deploy* contínuo em duas *Platform as a Service* (PaaS), por possuirem integrações com as *branches* do GitHub. Em ambas as plataformas *cloud*, é possível configurar para que ao atualizar uma *branch*, o seu código seja automaticamente implantado e disponibilizado via URI. Elas são:

- Heroku, para manter um banco de dados PostgreSQL, o código do *back-end* e subir as URLs das APIs HTTP, mantendo o servidor Rails rodando (James Lindenbaum, 2020). Possui uma conveniente interface para deploy automático de projetos Rails e uma forma integrada de construir um banco de dados PostgreSQL, por isso foi utilizado para manter o servidor do Scientiometer, durante as fases de desenvolvimento e teste.
- Netlify (Mathias Biilmann, 2020), para manter o código do *front-end*, subir as páginas gráficas e suas URLs para serem acessadas via navegador, mantendo o *runtime* Node.js (Tilkov e Vinoski, 2010). O Netlify pode implementar projetos Vue.js de forma automática e realizar a integração com a *branch* contendo seu código-fonte, então reimplementar o sistema de *front* do Scientiomenter, quando ocorre uma alteração do código-fonte.

3.1.3 Teste

Para garantir a funcionalidade das estruturas de um código, pode-se utilizar e manter uma série de testes automatizados. Esses testes devem ser executados a cada modificação no sistema, para verificar se essa alteração não quebrou os contratos firmados e abrangidos pelos testes.

O arcabouço (= *framework*) Rails já possui um conjunto de estruturas básicas para

testes de código, porém é conveniente adicionar outras gemas (= *gems*, nome técnico dado às bibliotecas Ruby autocontidas):

- RSpec-Rails (Rspec, 2013), para escrever o teste de forma descritiva em alto nível, quase puramente em inglês;
- Faker (Faker-Ruby, 2020), para criar objetos com dados aleatórios;
- Factory Bot (Thoughtbot, 2020), para estratégias de *build* de entidades (instâncias persistidas e não persistidas);
- Database Cleaner (Cleaner, 2020), para definir as estratégias de limpeza de banco de dados (truncar, apagar registros, estado de transação, entre outras).

Por um navegador *web*, o orientador acessou o sistema Scientiometer por sua interface do aplicativo *front-end*. Com a finalidade de testar o ambiente, ele utilizou diferentes contas de usuários fictícios e com diferentes papéis dentro da organização (pesquisador, diretor de laboratório e diretor de divisão), pois esses papéis definem quais do Scientiometer podem ser acessados. Em cada momento, era simulado um preenchimento parcial de um relatório de produtividade, desde o passo 1 até o passo a ser testado naquela rodada de testes. Esses testes também eram acompanhados pelo orientando. Ao identificarem *bugs*, inconsistências ou melhorias desejadas, esses itens eram anotados na ferramenta Trello (descrito em 3.1.5).

3.1.4 Estudo da estrutura do sistema

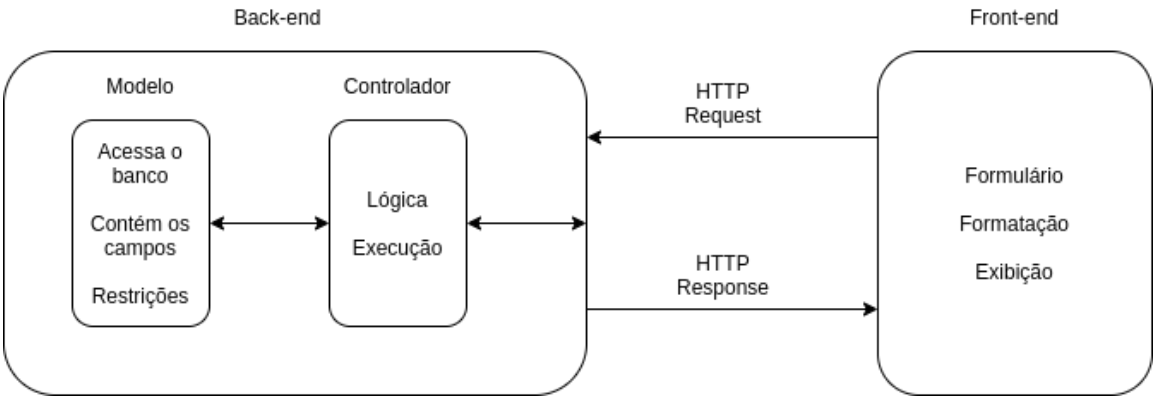


Figura 3.4: *Arquitetura* Model-View-Controller *(MVC) (Krasner, Pope et al., 1988) utilizada para construir o sistema Scientiometer e suas principais entidades.*

O Scientiometer foi construído sob o modelo MVC (ver figura 3.4). MVC é uma arquitetura de software baseada em três diferentes componentes funcionais:

- *model*, os diferentes objetos do domínio de aplicação (no caso do Scientiometer: pesquisadores, laboratórios, relatórios de produtividade etc). No Scientiometer, são entidades *class* e herdam da classe *ApplicationRecord*. Ao herdar dessa classe, o modelo é mapeado com a tabela de mesmo nome no banco de dados e operações sobre essa tabela podem ser efetuadas por chamadas de métodos;

- *view*, a camada de apresentação, responsável por lidar com toda a parte gráfica do sistema (e.g. páginas, botões, formulários). No Scientiometer, essa funcionalidade é feita pelo aplicativo de *front-end*. O *back-end* e o *front-end* trocam dados, via requisições HTTP, para que o *front-end* possa construir a visualização necessária ao usuário corrente;
- *controller*, responsável pelo processamento das interações entre o usuário e o sistema. No Scientiometer, são entidades *class* e herdam da classe *ApplicationController*. Ao herdar dessa classe, são definidos métodos úteis ao processamento de requisições, como filtragem de parâmetros recebidos via URL ou *payload*. Herdar de *ApplicationController* também faz com que os métodos (=*actions*) definidos nessa classe possam ser mapeados pelo roteador.

3.1.5 Controle do fluxo de trabalho

Como estratégia de trabalho e organização das funcionalidades a serem reparadas ou desenvolvidas, utilizou-se a metodologia Kanban (Huang e Kusiak, 1996), baseada em um painel com cartões, cada um desses possuindo uma descrição do item a ser feito e sua data limite. O Trello foi o *software* usado para essa prática. Ele é uma aplicação *web* que contém um *board* e podem ser colocados *cards* com título, descrição, *flags* coloridas, responsável, data limite e outros. O programa também é colaborativo, possuindo funcionalidades para times e divisões e subdivisões de projetos.

3.2 Refatoração do sistema

A refatoração do sistema Scientiometer foi concentrada na resolução do fluxo de criação do relatório de produtividade. Esse fluxo está contido no arquivo de controlador *forms_controller.rb*. Isso quer dizer que os *endpoints* no *back-end* de recebimento (verbo POST de HTTP) do relatório de produtividade foram os mais modificados.

```
-           grant.cnpq_level_id = entry[:cnpq_level_id][:value]
+           grant.cnpq_level_id = entry[:cnpq_level][:value]
```

Figura 3.5: *Exemplo de correção efetuada no* back-end *do sistema Scientiometer: o nome do parâmetro recebido estava incorreto, portanto, o passo de um relatório dessa entidade não estava persistindo no banco de dados. Houve a renomeação adequada para salvar um registro do passo 2 do relatório e um registro de bolsa de produtividade. Figura feita com a interface gráfica do GitHub, que implementa a ferramenta* diff *de comparação de arquivos. Essa modificação é parte da solução para o erro apresentado na figura 3.3.*

No *front-end*, em geral, houve necessidade de melhorias na visualização de dados já inseridos no relatório. Como na figura 3.6, os dados de um funcionário inserido no relatório de produtividade não podiam ser vistos.

Para garantir o funcionamento do sistema e sua robustez futura, foram adicionados testes unitários. Esses testes são trechos de códigos que executam outras partes do código-fonte e comparam um resultado esperado com o resultado obtido em sua execução.

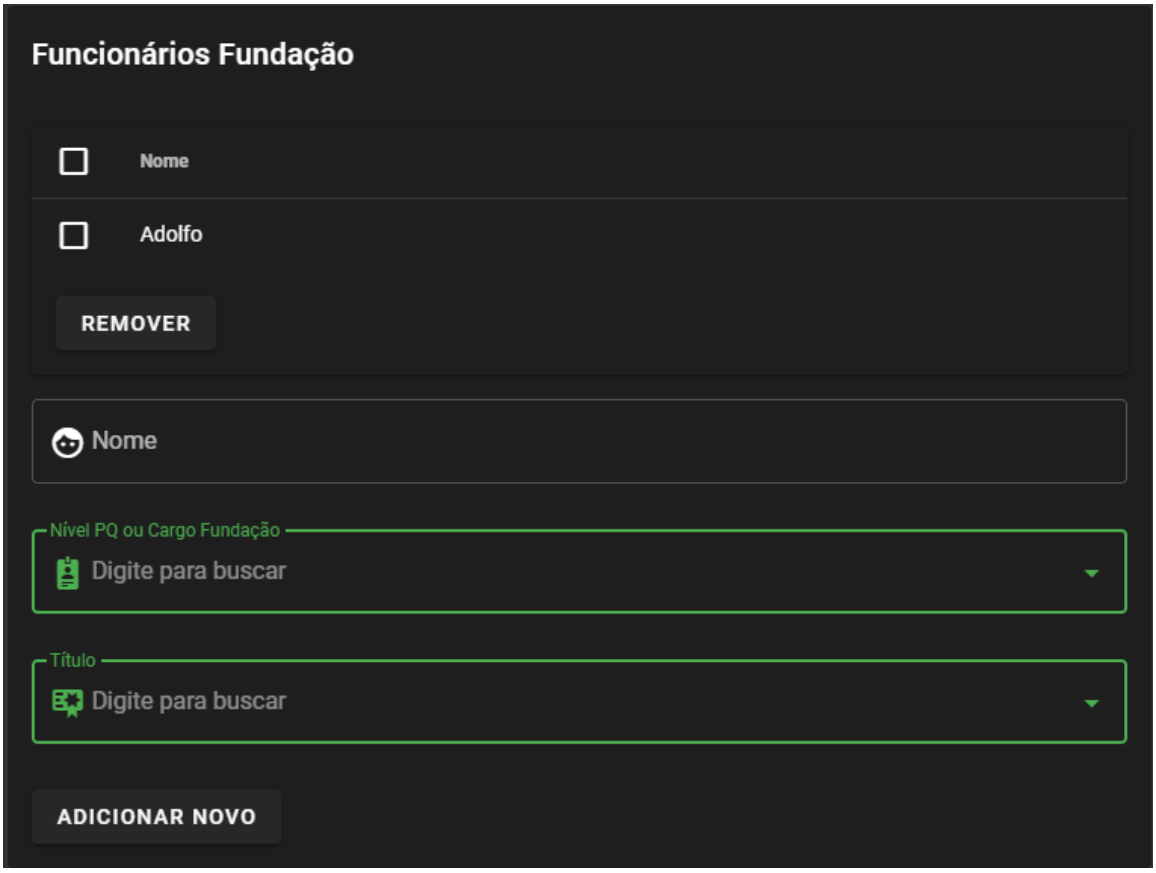
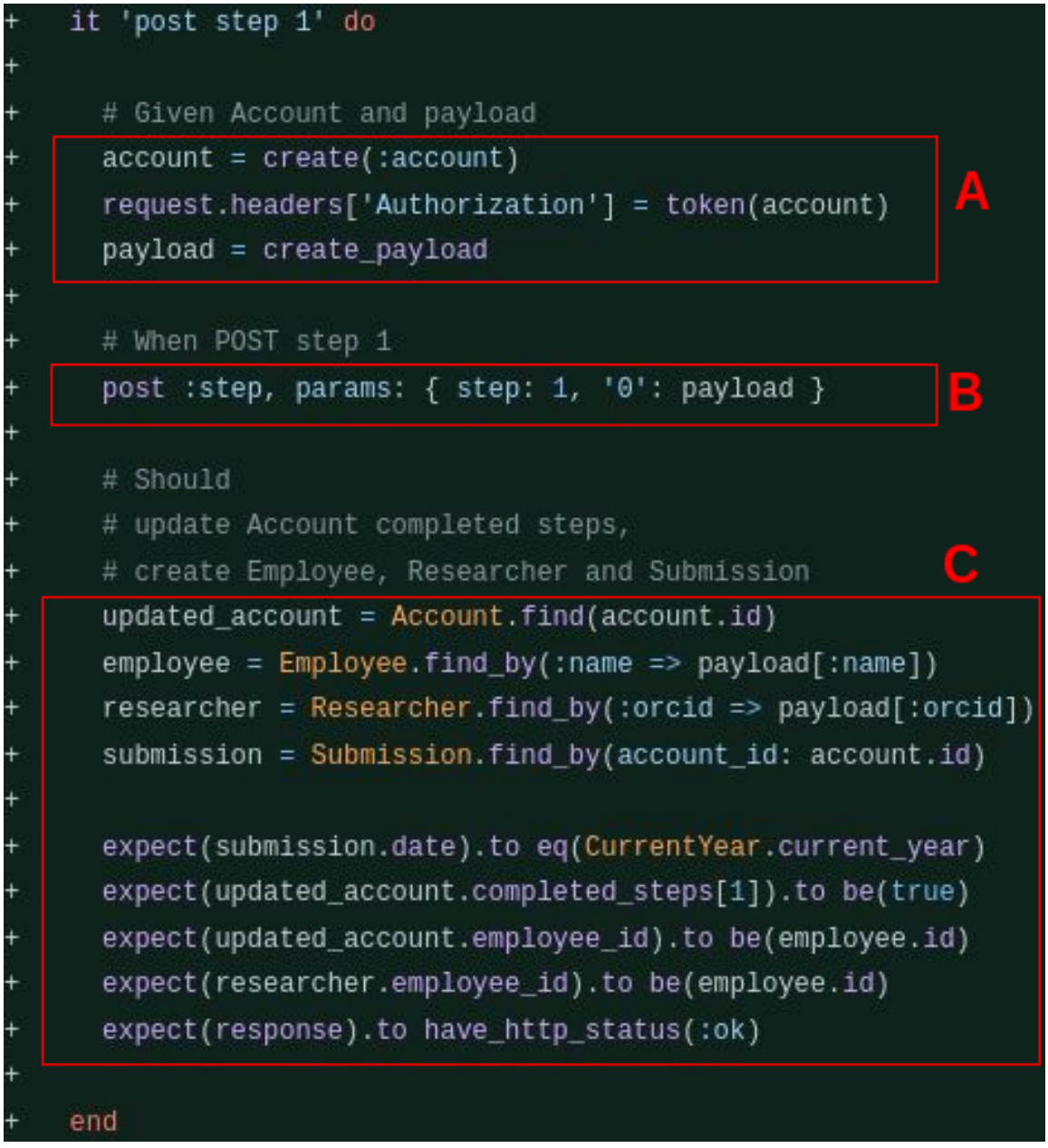


Figura 3.6: *Passo 2 do relatório de produtividade científica, exemplificando como o Scientiometer não possuía tabelas completas das entidades adicionadas ao relatório, tais como os dados de funcionários do IB.*

Foi criada uma abstração *Service*, com a finalidade de aumentar a modularização e a segregação de responsabilidades. O sistema Scientiomter possuía o *controller* com muitas atribuições, pois recebia a requisição, processava os dados e retornava o resultado construído.

A nova funcionalidade de serviço tem a incumbência desse processamento, deixando para o controlador a tarefa de determinar qual parte do sistema é responsável por tratar a requisição *web* recebida e abastecer a resposta com o retorno do serviço específico.




```
+   it 'post step 1' do
+
+     # Given Account and payload
+     account = create(:account)
+     request.headers['Authorization'] = token(account)
+     payload = create_payload
+
+     # When POST step 1
+     post :step, params: { step: 1, '0': payload }
+
+     # Should
+     # update Account completed steps,
+     # create Employee, Researcher and Submission
+     updated_account = Account.find(account.id)
+     employee = Employee.find_by(:name => payload[:name])
+     researcher = Researcher.find_by(:orcid => payload[:orcid])
+     submission = Submission.find_by(account_id: account.id)
+
+     expect(submission.date).to eq(CurrentYear.current_year)
+     expect(updated_account.completed_steps[1]).to be(true)
+     expect(updated_account.employee_id).to be(employee.id)
+     expect(researcher.employee_id).to be(employee.id)
+     expect(response).to have_http_status(:ok)
+
+   end
```

Figura 3.7: *Um teste unitário criado no* back-end *do sistema Scientiometer visando testar e garantir o funcionamento da criação do passo 1 do relatório de produtividade científica. O teste é dividido em 3 partes:* A*, preparação inicial dos dados, nesse caso, é necessário existir uma conta de um usuário, um* token *de autenticação e um conjunto de dados a ser colocado no* payload *da requisição HTTP com o verbo POST;* B*, ação que deve disparar o trecho de código a ser testado, nesse caso, uma requisição HTTP com verbo POST irá executar o trecho de código referente ao passo 1 de criação do formulário de produtividade científica contido no arquivo* forms_controller.rb*;* C*, comparações para-a-par entre resultados obtidos (contidos dentro das chamadas da função* expect*) e os resultados esperados (contidos dentro das chamadas da função* be *ou outras semelhantes). Esse teste está no arquivo* spec/controllers/forms_controller.rb.

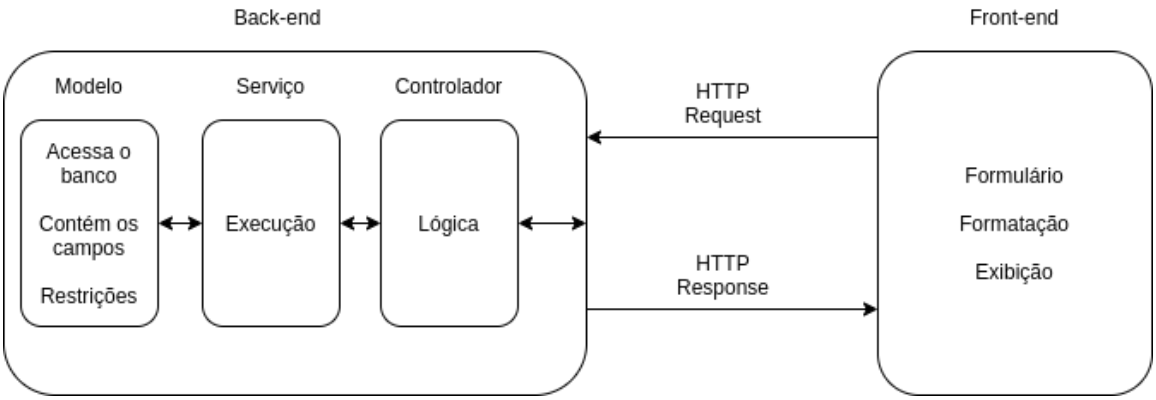


Figura 3.8: *Arquitetura MVC estendida utilizada no Scientiometer para contemplar a nova camada de abstração* Service.

```
-       employee = Employee.new
-       employee.name = filt_params[:name]
-       employee.is_foundation = true
-       employee.laboratory_id = filt_params[:laboratory_id]
-       employee.role_foundation_level_id = filt_params[:role_foundation_level_id]
-       employee.title_id = filt_params[:title_id]
-
-       employee.save!
-
-       researcher = Researcher.new
-       researcher.researcher_id = filt_params[:researcher_id]
-       researcher.ORCID = filt_params[:orcid]
-       researcher.email = filt_params[:email]
-       researcher.employee_id = employee[:id]
-       researcher.post_doc_type_id = filt_params[:post_doc_type_id]
-       researcher.ingress_date = filt_params[:ingress_date]
-
-       researcher.save!
+       employee = EmployeeService.create_employee(filt_params)
+       ResearcherService.create_researcher(filt_params, employee.id)
```

Figura 3.9: *Refatoração de trecho do passo 1 do relatório de produtividade no* back-end *do sistema Scientiometer. A responsabilidade de criação das entidade Funcionário e Pesquisador foram removidas do controlador* forms_controller.rb*, que deve ser responsável por direcionar os dados aos serviços específicos de CRUD. A criação explícita, naquele arquivo, foi substituída pelas chamadas das funções recém-criadas.*

Capítulo 4

Resultados

Neste capítulo são apresentados os resultados obtidos após o processo de refatoração do Scientiometer, detalhado no capítulo 3. Inicialmente, são apresentadas instâncias das inconsistências que foram corrigidas. Por fim, é tratada a implantação do sistema em uma servidora.

4.1 Instâncias de inconsistências corrigidas

Nesta seção, há exemplos de inconsistências que foram corrigidas: a correção em salvar um relatório de produtividade que foi preenchido corretamente; visualizar um relatório já iniciado; adicionar novos dados a um relatório; fluxo de trabalho com submissão, aprovação, reprovação e comentários; melhor visualização dos dados do relatório e, por fim, painel de usuário com alteração de senha. A refatoração do sistema Scientiometer teve melhorias e adequações nos fluxos de processo pela utilização dos usuários (comparar figuras 3.1 e 4.1) e são explicados a seguir.

4.1.1 Preenchimento de relatório

Anteriormente, o preenchimento do formulário de produtividade não poderia ser completado: havia erros e bugs que impediam o prosseguimento dos passos (ver figura 4.2). Após a refatoração, todos os passos podem ser preenchidos e salvos pelos pesquisadores (ver figura 4.3).

4.1.2 Visualização de relatório já iniciado

Anteriormente, um pesquisador não conseguia visualizar um relatório seu já iniciado, seja esse concluído ou apenas preenchimento parcial. Após a refatoração, é possível um pesquisador visualizar um relatório seu que já esteja sido iniciado ou finalizado, como pode ser visto nas figuras 4.4 e 4.5.

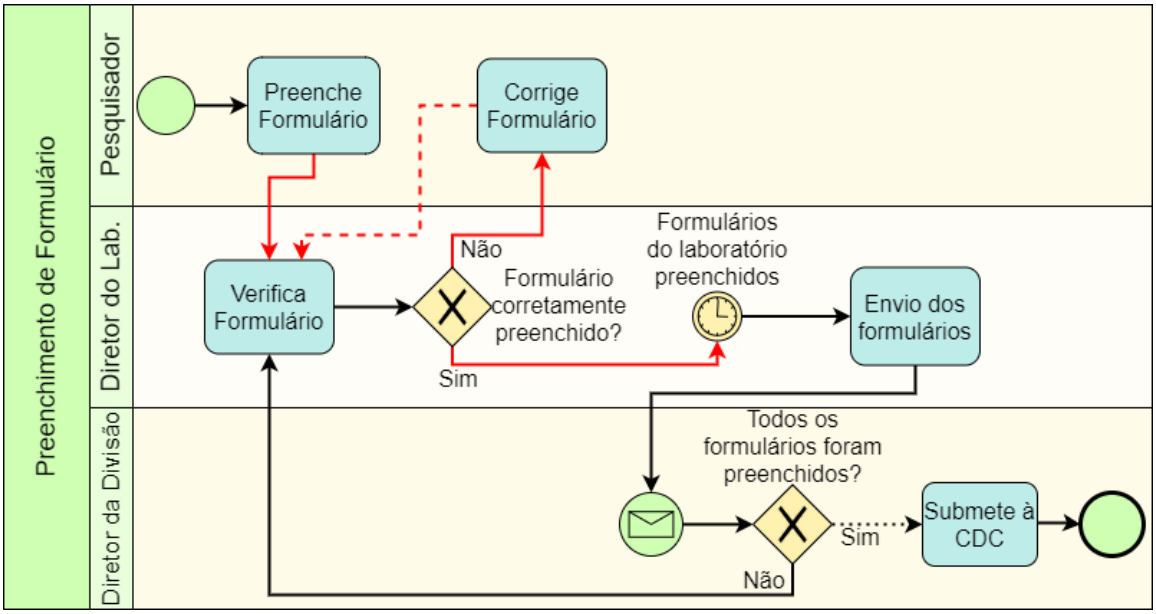


Figura 4.1: *BPMN do sistema Scientiometer após a refatoração. Em vermelho, os fluxos com as contribuições mais relevantes para o trabalho.*

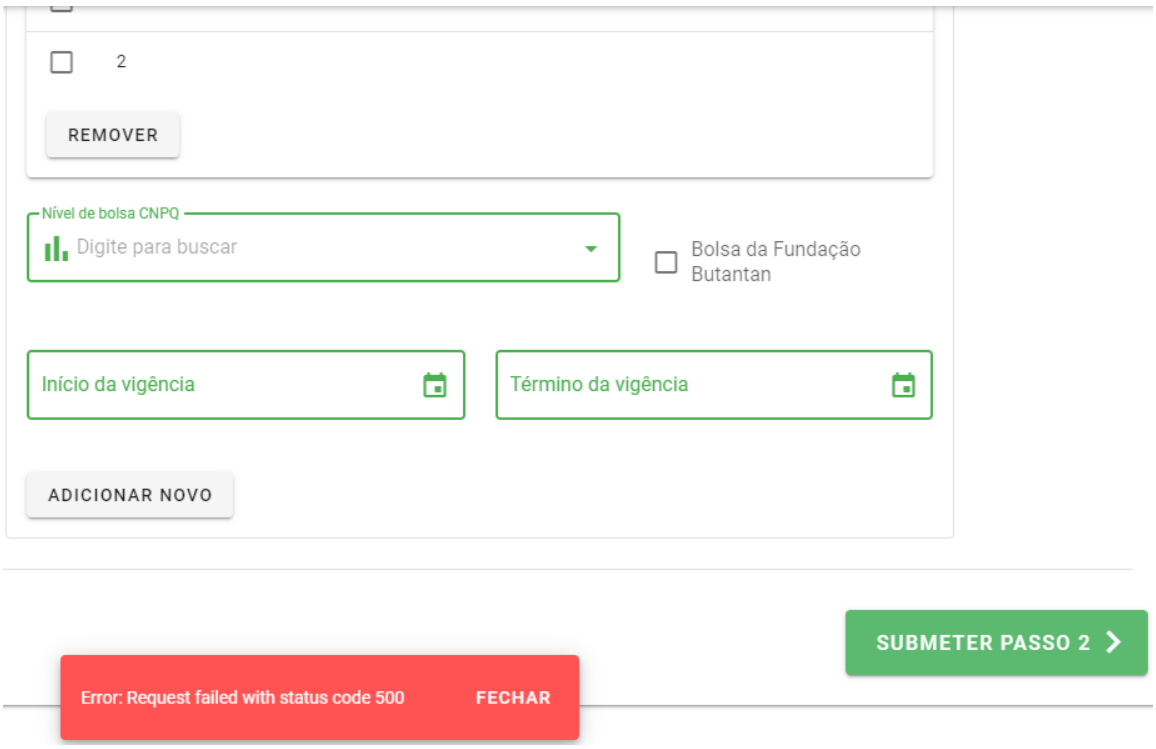


Figura 4.2: *Erro em preencimento de relatório de produtividade, mostrando que antes da refatoração, havia erros ao tentar salvar algum passo.*

4.1.3 Adição de dados em relatório já iniciado

Anteriormente, um pesquisador não conseguia interagir com um relatório seu já iniciado.

Figura 4.3: *Sucesso em preencimento de relatório de produtividade, mostrando que após a refatoração, os passos podem ser salvos.*

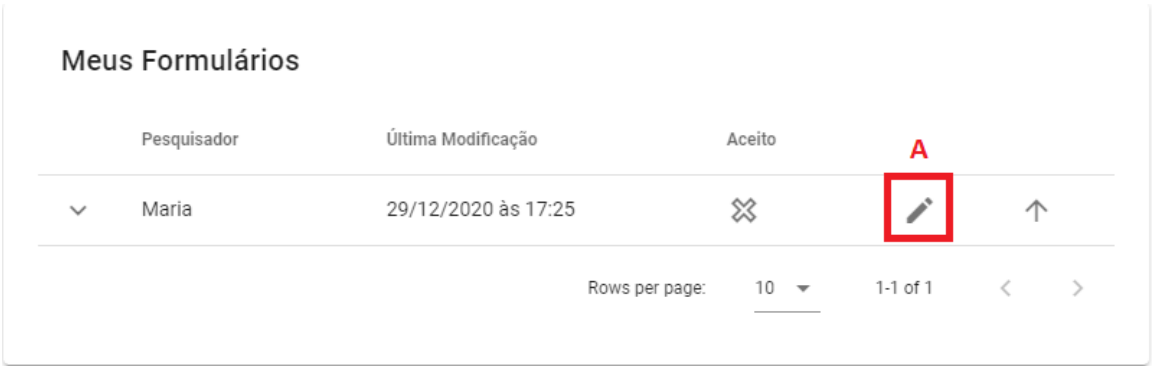


Figura 4.4: *Tela de Minhas Submissões. Ao clicar em A, o relatório iniciado anteriormente poderá ser visualizado por seu autor (ver figura 4.5).*

Após a refatoração, é possível os pesquisadores adicionarem novos dados em um relatório já iniciado e ainda não aprovado, como pode ser visto nas figuras 4.6–4.8.

4.1.4 Fluxo de aprovação

Foi criado novo fluxo de trabalho para visualização de formulário já preenchido (figura 4.9), submissão (figura 4.11), aprovação (figura 4.14), reprovação de formulário (figura 4.13)

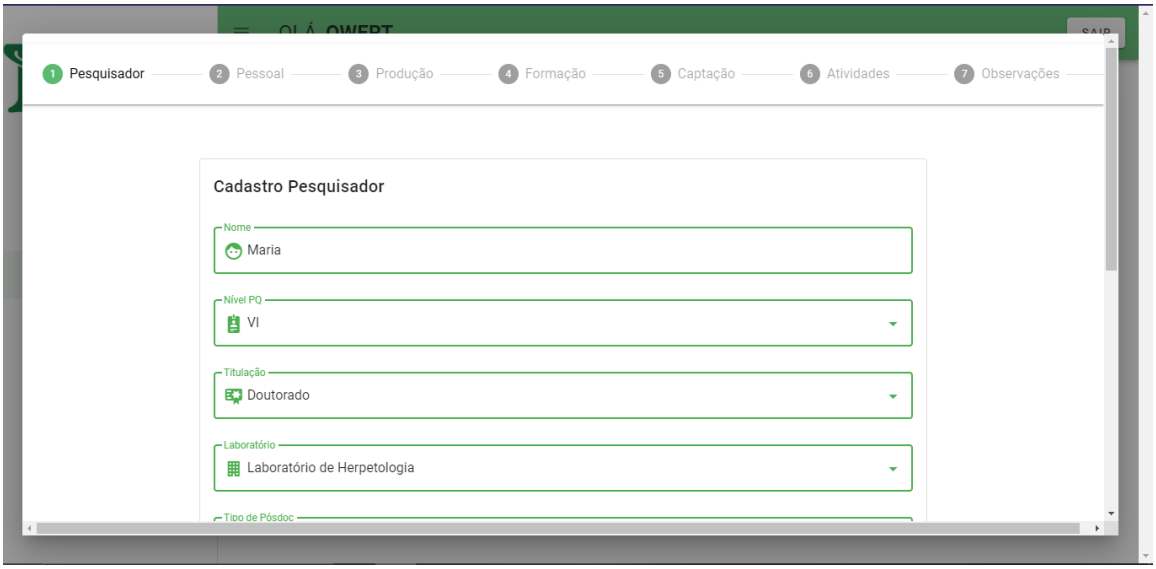


Figura 4.5: *Tela de Minhas Submissões. Visualização de um relatório já iniciado (ver figura 4.4).*

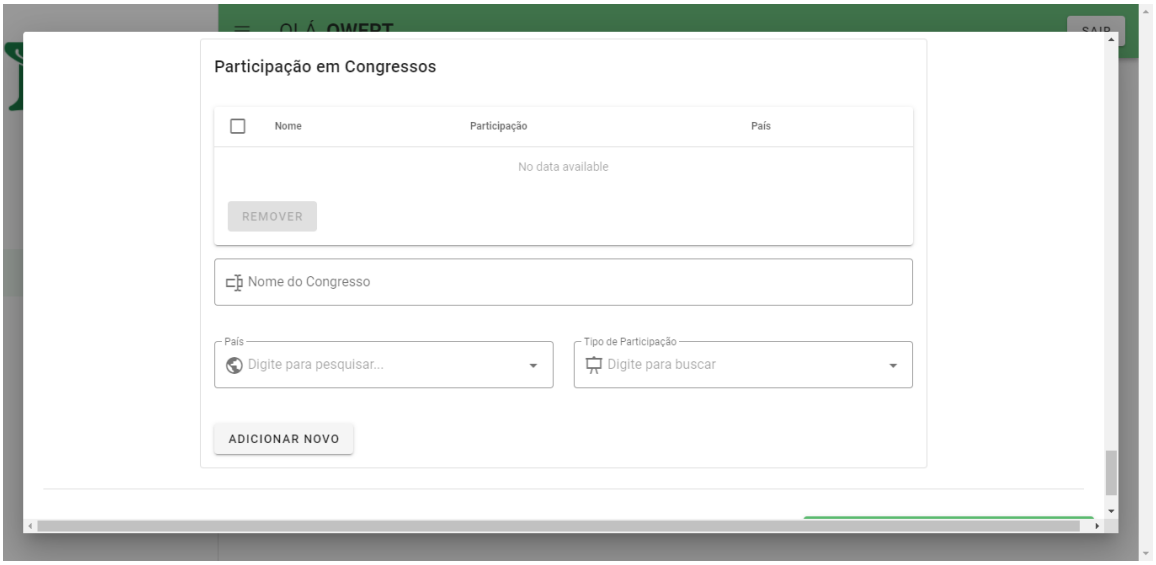


Figura 4.6: *Tela de Minhas Submissões. Um relatório já iniciado sem registros em um passo qualquer.*

e comentários (figura 4.16).

4.1.5 Melhoria na tabela de visualização de dados

Após modificar o *back-end*, para incluir certos campos no *payload* de resposta dos dados requisitados pelo *front-end*, foi possível adicionar mais dados às tabelas de visualização de entidades do relatório de produtividade. A figura 4.17 mostra uma das melhorias realizadas.

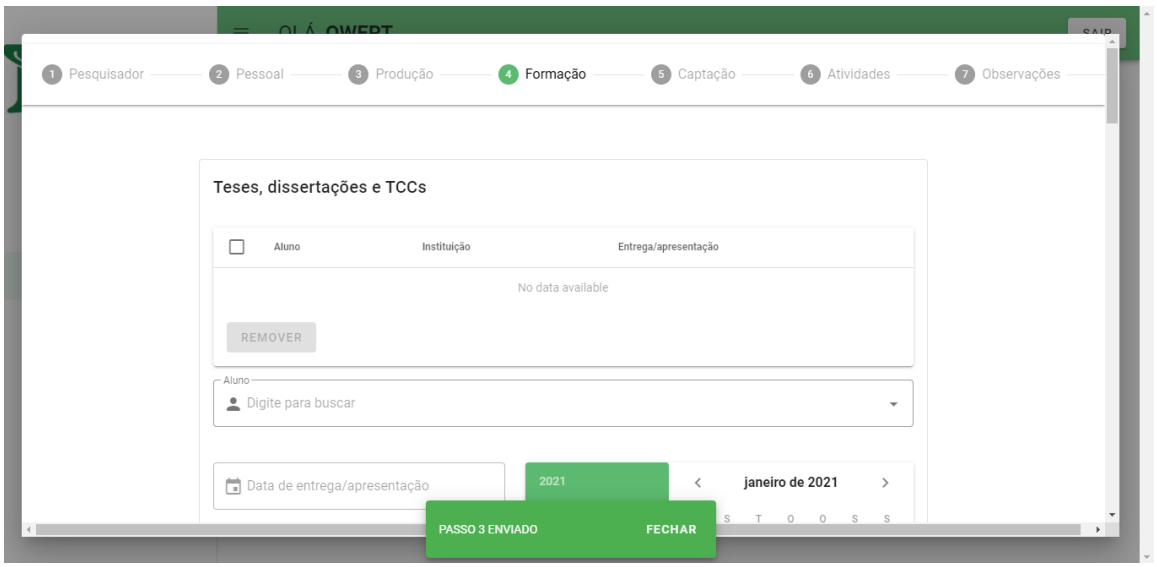


Figura 4.7: *Tela de Minhas Submissões. Um relatório já iniciado com mensagem de sucesso ao adicionar mais dados.*

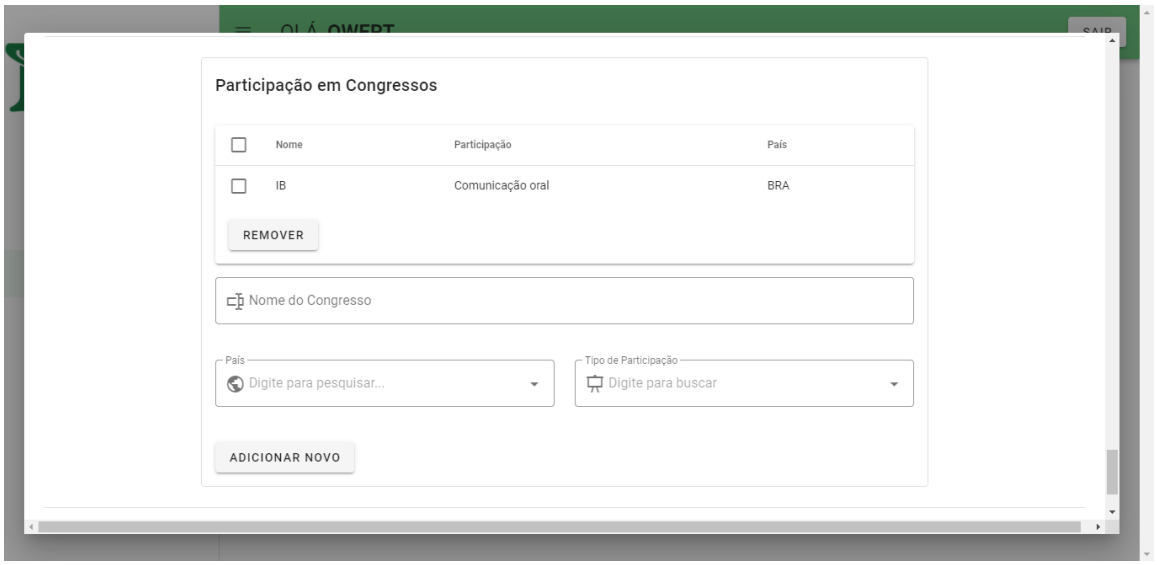


Figura 4.8: *Tela de Minhas Submissões. Um relatório já iniciado com dados que foram adicionar após início em outro momento.*

4.1.6 Painel de usuário

Foi criada página de painel de usuário, com *link* localizado na barra lateral. Nela, é possível verificar dado de usuário da conta de acesso e realizar alteração de senha (ver figura 4.8). A página pode acomodar mais dados relacionados à conta ou ao pesquisador.

4.2 Cobertura de testes

Cobertura de testes é o percentual de linhas de código executadas ao menos uma vez nos testes unitários; é uma medida objetiva de qualidade e adequação do *software* (Zhu

et al., 1997).

Para calcular a cobertura de testes durante essa refatoração do sistema Scientiometer(ver 3.1.3), utilizou-se a *gem* simplecov (Simplecov-Ruby, 2020). Essa gema é uma biblioteca que calcula quantas linhas de código foram executadas pelos testes unitários e também provê uma interface gráfica para visualizar os resultados.

Figura 4.9: *Barra lateral de tarefas do sistema Scientiometer. Após a refatoração, agora ela possui opção para visualizar um relatório preenchido.*

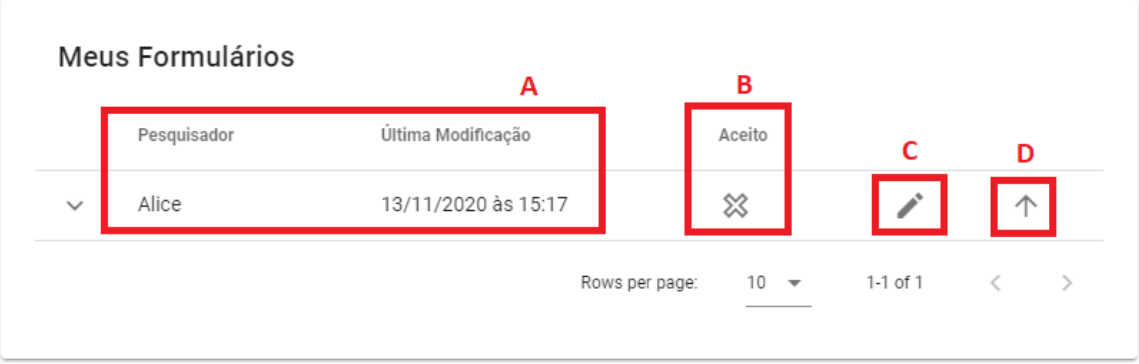


Figura 4.10: *Tela de Minhas Submissões. Nela, o pesquisador pode visualizar e interagir com um relatório já preenchido. (A) informações de autoria e data da modificação mais recente. (B) status de aprovação, nesse caso, ainda não foi aprovado e não está submetido à análise. (C) visualizar ou adicionar dados de produtividade ao relatório. (D) submeter relatório à análise ao seu diretor de laboratório.*

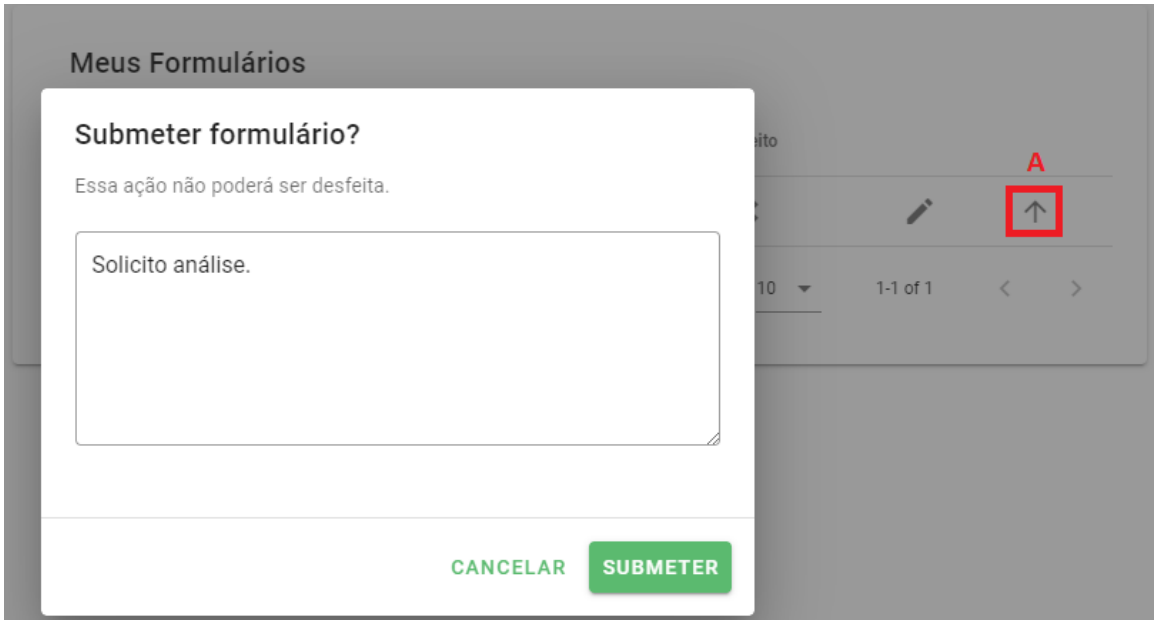


Figura 4.11: *Tela de Minhas Submissões e* card *de submissão. Ao clicar no ícone* A*, uma caixa de aviso é aberta ao pesquisador, que pode inserir uma mensagem opcional e clicar em* SUBMETER*. Se realizar essa ação, o formulário poderá ser aprovado ou reprovado pelo diretor do laboratório do autor do formulário de produtividade.*

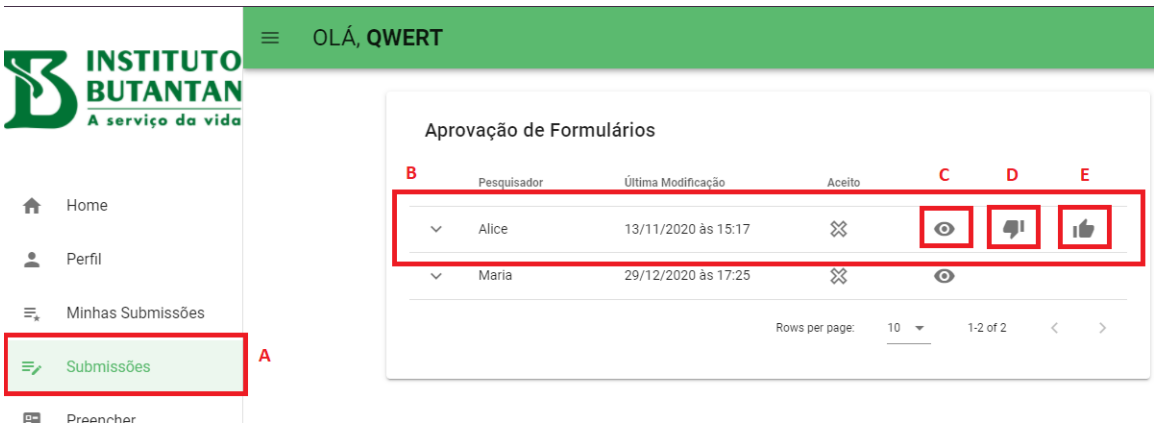


Figura 4.12: *Tela de Submissões, na visão de um diretor de laboratório. Um diretor de laboratório pode acessar o menu de Submissões* (A) *e interagir com relatórios de produtividade de pesquisadores de seu respectivo laboratório. Nesse exemplo, a pesquisadora Ana, da sequência de 4.11, submeteu seu relatório de produtividade à apreciação. O relatório de Ana* (B) *então está disponível aos diretores para visualizão* (C)*, reprovação* (D) *ou aprovação* (E)*. Note que o relatório de Maria não pode ser aprovado ou reprovado, pois ainda não foi submetido por sua autora. Nesse exemplo, estamos na visão de Maria, que é a diretora capaz de aprovar o relatório de Ana e o seu próprio, o que significa que o fluxo de submissão deve ser seguido por todos os cargos hierárquicos.*

O controlador mais importante do ponto de vista de persistência de dados está no arquivo *forms_controller.rb*, pois ele contém o código para registrar no banco de dados todos os registros contidos nos relatórios de produtividade dos pesquisadores que utilizam o sistema Scientiometer. O teste de cobertura para esse arquivo passou de 0% para 100% (pode ser visto na figura 4.19).

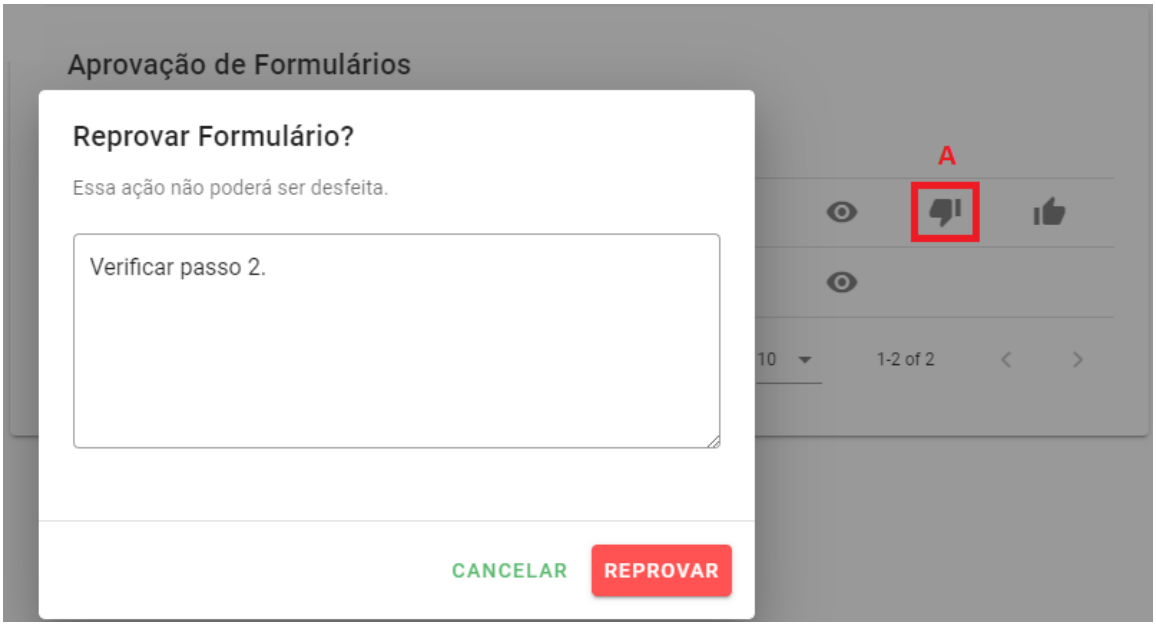


Figura 4.13: *Tela de Submissões e* card *de reprovação. Um diretor de laboratório ao clicar em* A *poderá deixar uma mensagem opcional e clicar em* REPROVAR*, para que o pesquisador verifique o seu relatório de produtividade.*

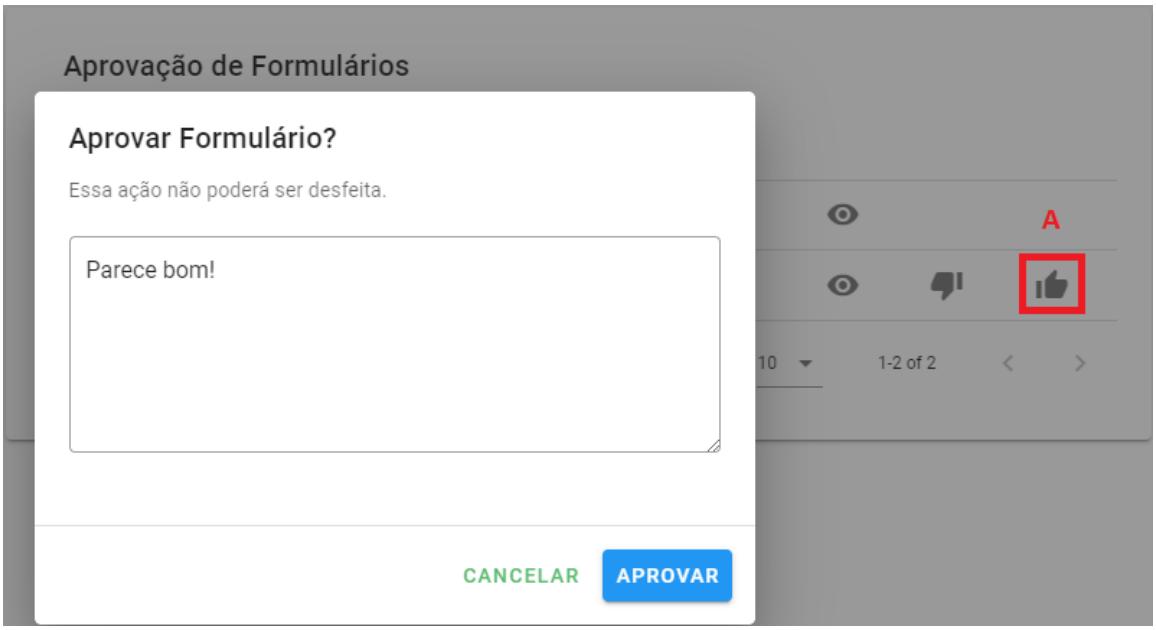


Figura 4.14: *Tela de Submissões e* card *de aprovação. Um diretor de laboratório ao clicar em* A *poderá deixar uma mensagem opcional e clicar em* APROVAR*, para aprovar o relatório de produtividade. No exemplo atual, o relatório havia sido reprovado (4.13) e, novamente, submetido à analise, por Ana, autora do relatório. Esse ciclo de submissão e reprovação poderá ser feito quantas vezes forem necessárias até a aprovação, que é final.*

Utilizando o comando *rspec*, na raíz do projeto, será criado um diretório *coverage/*, contendo os metadados dos testes e o arquivo *coverage/index.html*, utilizado para navegar nos diferentes arquivos e verificar suas coberturas de código.

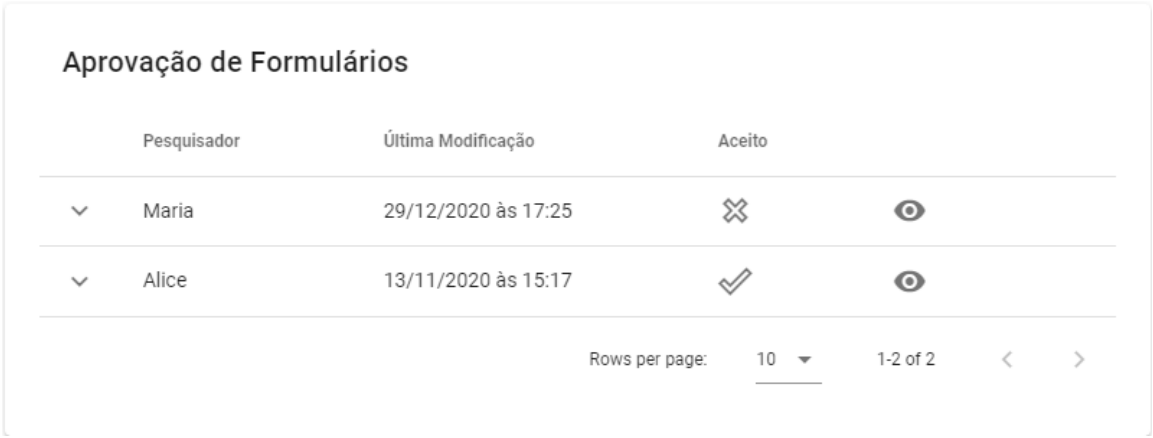


Figura 4.15: *Tela de Submissões contendo um relatório aprovado. Após aprovação de um relatório de produtividade (identificado por um* check *na coluna* Aceito*), ele não mais poderá ser alterado por seu autor e nem ter seu status de aprovação modificado por um diretor; o autor e seus diretores ainda poderão visualizar o relatório aprovado (clicando no símbolo de olho).*

Figura 4.16: *Tela de Minhas Submissões e tabela de aprovação. Durante todas as fases de aprovação de um relatório de produtividade, o autor do relatório e seus diretores podem clicar em* A *para acompanharem as ações realizadas, seus agentes, datas e os comentários opcionais.*

4.3 Implantação do sistema refatorado

Implantação (*deploy*) de um sistema de informação é a sua instalação, usualmente em servidora, e configuração visando disponibilizar esse sistema para uso em produção por parte dos usuários.

4.3.1 Implantação anterior

Uma implantação parcial do Scientiometer antes de sua refatoração foi feita pelo aluno, com a participação de David Pires (administrador de sistemas do Laboratório de Ciclo Celular do IB) e de Bruno Scholl (desenvolvedor original do sistema).

O aplicativo de *back-end* utilizando o *framework* Rails foi implementado em uma

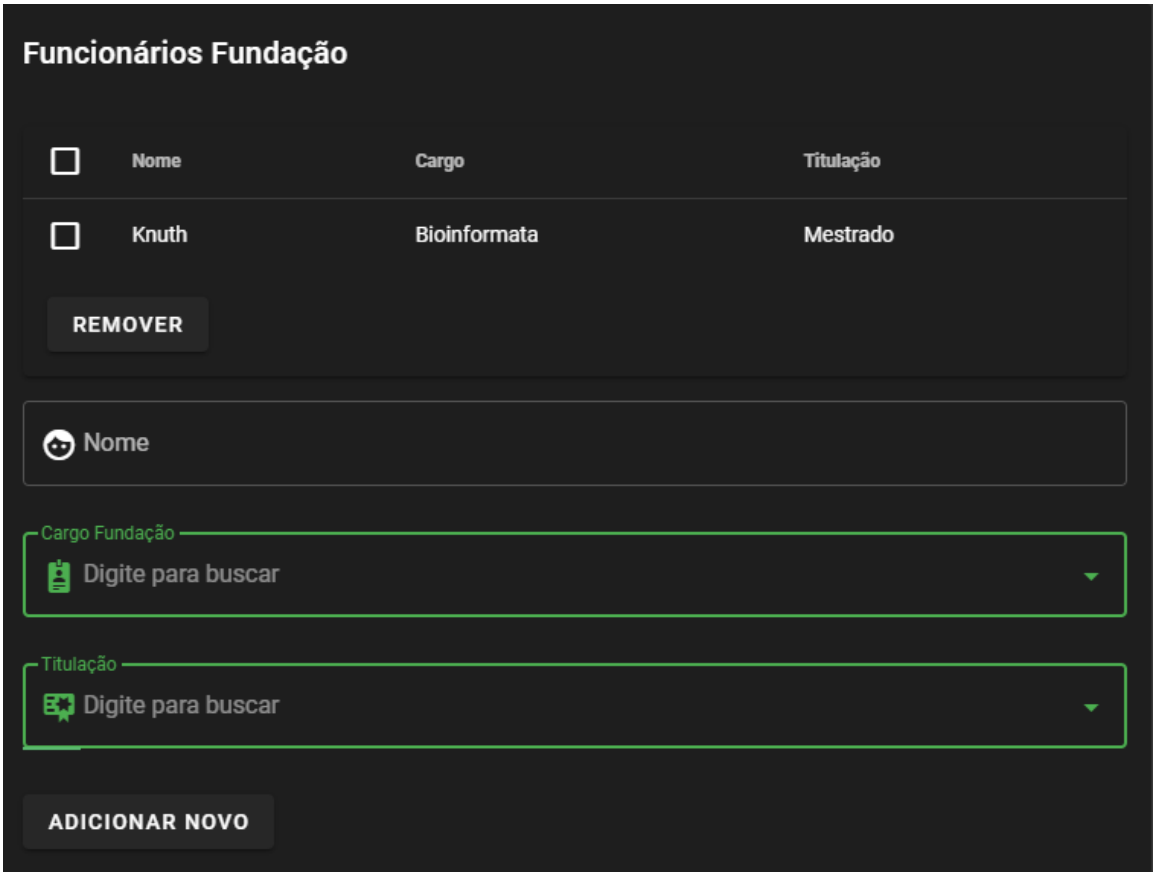


Figura 4.17: *Passo 2 do relatório de produtividade científica, após melhorias para visualização dos dados na tabela de funcionários do IB que foram adicionados ao relatório de produtividade no sistema Scientiometer.*

máquina servidora local do IB. Foi necessário configurar arquivos do Apache HTTP Server (Robert McCool, 2020), um software de servidor web de plataforma cruzada gratuito e de código aberto, contendo dados de caminho do diretório raiz do projeto, porta e permissões de execução e redirecionamento do servidor.

O *front-end* não foi implementado, por razões técnicas não solucionadas. Devido ao agravamento da pandemia de COVID-19, interromperam-se as tentativas e o aluno buscou prosseguir com outra etapa no projeto Scientiometer. Dessa forma, o sistema Scientiometer ainda não está disponível para ser utilizado ou testado em produção.

4.3.2 Implantação futura

Atualmente, o sistema encontra-se em estágio final de refatoração e testes e, quando essas tarefas forem terminadas, uma nova implantação deverá ser feita, só que desta vez remotamente devido à pandemia de COVID-19. O sistema deverá ficar em servidora local do IB com os aplicativos de *back-end*, *front-end* e banco de dados PostgreSQL, para que possa ser testado e utilizado por usuários finais.

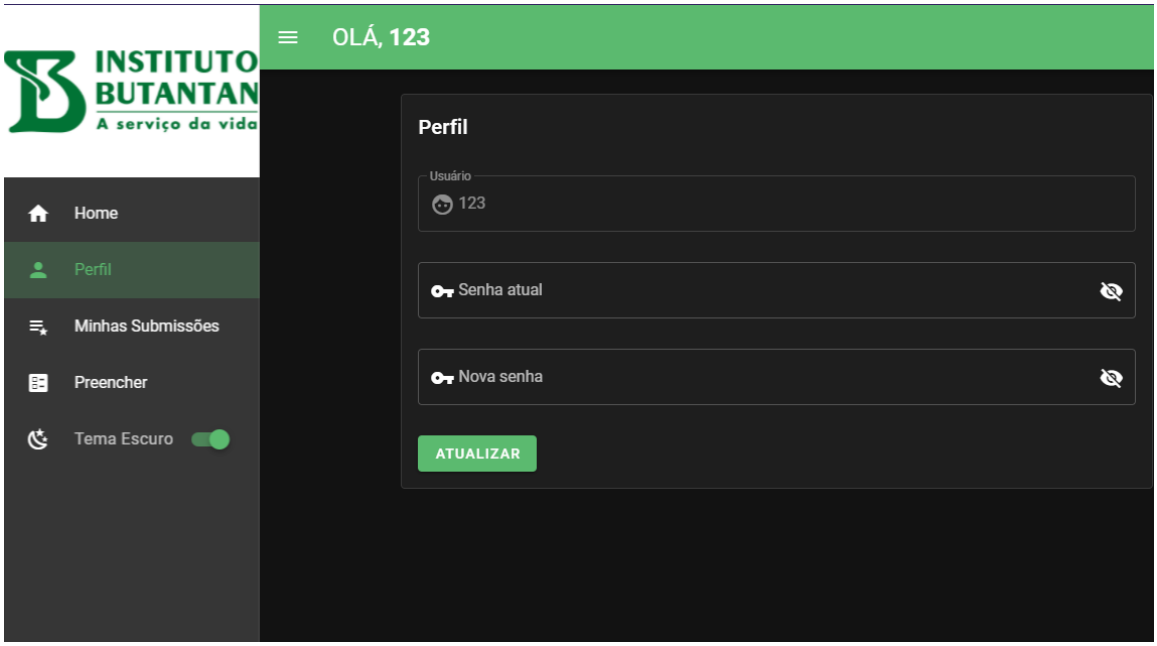


Figura 4.18: *Tela de Perfil. Após a refatoração, existe uma página de painel de usuário, onde é possível alterar a senha da conta de acesso ao sistema Scientiometer.*

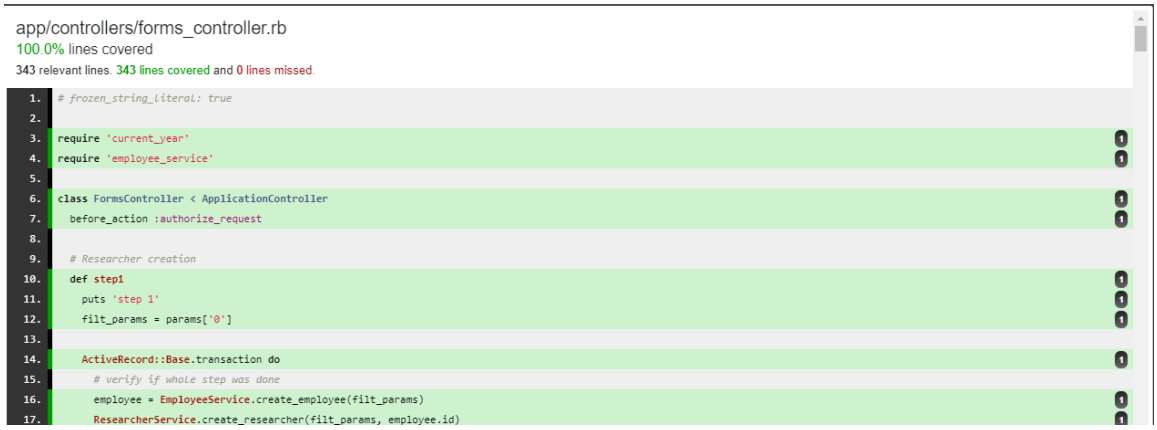


Figura 4.19: *Relatório de cobertura de código do sistema Scientiometer com utilização da biblioteca* simplecov. *O controlador responsável pela criação do relatório de produtividade agora possui 100% de cobertura de testes.*

Capítulo 5

Conclusão

Neste capítulo é recapitulado o conteúdo deste trabalho, destacando-se as principais contribuições. Além disso, são descritos trabalhos futuros interessantes para a melhoria e ampliação do que foi desenvolvido.

5.1 Recapitulação

O sistema de inserção de dados de produtividade científica Scientiometer necessitava de correções de *bugs* e inconsistências funcionais, além de melhorias internas de código fonte e usabilidade pelos pesquisadores do IB. Sintetizando o que foi mostrado nos capítulos anteriores, listam-se:

- não era possível avançar durante os passos de inserção de dados, por erros não sanáveis pelo usuário;
- impossibilidade de prosseguir, em outro momento, de um relatório já iniciado;
- um pesquisador não conseguia ver o relatório que já preencheu, parcial ou totalmente;
- as tabelas dos dados de produtividade inseridos não continham todos os valores necessários para plena conferência de consistência;
- não havia fluxo de reprovação de relatório;
- um relatório iniciado não poderia retornar a um pesquisador para realizar inserções ou correções;
- não havia maneira explícita de comunicação interna ao sistema para que um diretor indicasse um motivo de aprovação ou reprovação de relatório.

Foi feita uma refatoração do sistema a partir de técnicas de refatoração, que buscam reconhecer padrões de maus hábitos e solucionar esses problemas. Criou-se uma cobertura de testes que auxiliará os próximos desenvolvedores a garantir a funcionalidade necessária ao sistema. Além disso: foram corrigidos *bugs*; implementada página para visualizar relatórios (em andamento e finalizados) com seus status e botões e caixas de texto para interação com fluxo de aprovação, com tabela das ações e os comentários dos usuários;

página de painel de usuário e melhorias visuais nas tabelas de dados de produtividade do relatório.

Essa refatoração obteve resultados, pois agora é possível criar, modificar (ver limitação na seção 5.2.1), submeter, visualizar, aprovar e reprovar os relatórios de produtividade, além de melhor verificação, pelo pesquisador, da consistência dos dados inseridos no relatório e painel de usuário.

5.2 Projetos futuros

Além da finalização dos testes e da implantação do sistema mencionados no capítulo anterior, nesta seção são sugeridos quais são os pontos de melhorias ainda necessários ao sistema Scientiometer, em particular a inclusão de novas funcionalidades.

5.2.1 Alteração de dados no relatório de produtividade

Nesse trabalho, foi desenvolvida a funcionalidade de salvar e adicionar novos dados de produtividade a um relatório. Ainda resta modificar a funcionalidade de alteração de um dado já inserido, pois não há como modificar ou remover (ver figura 4.1; o fluxo de correção de formulário está parcialmente implementado).

Como sugestão, faz-se preciso modularizar ainda mais o controlador de recebimento dos dados do *front-end*, de forma a implementar os verbos HTTP *PUT* (alteração) e *DELETE* (remoção). Essa tarefa pode ser amparada e refeita com segurança, pois a criação (verbo *POST*) está com cobertura de testes em 100% (ver seção 4.2).

No *front-end*, precisa-se modificar as tabelas de apresentação dos dados, para que seja possível alterar os dados de um registro ou apagá-lo permanentemente do formulário. Atualmente, a remoção da tabela funciona para o caso de um novo dado ainda não persistido, porém um registro já persistido não é removido do banco de dados.

5.2.2 Análise de dados

Para ajudar a visualização de dados agregados e auxiliar a tomada de decisão baseada em produtividade científica, uma possibilidade interessante é a criação de um módulo de análise de dados. Com isso, pode-se gerar de forma automatizada tabelas, gráficos e outros elementos visuais que atualmente são produzidos de forma manual. Além disso, também é possível automatizar a criação de gráficos de análise individual dos pesquisadores.

No início deste trabalho, antes de ser colocada como prioritária a refatoração do sistema, foram testadas algumas opções de geração desses gráficos, como por exemplo, gráficos de barras, de pizza e de camadas para avaliar número de orientações de pesquisadores por vínculo empregatício dos mesmos (figuras 5.1–5.3). Esses propótipos de gráficos automatizados encontram-se disponíveis juntamente com o código refatorado do Scientiometer, podendo então ser facilmente aproveitados para uma eventual construção do módulo de análise de dados.

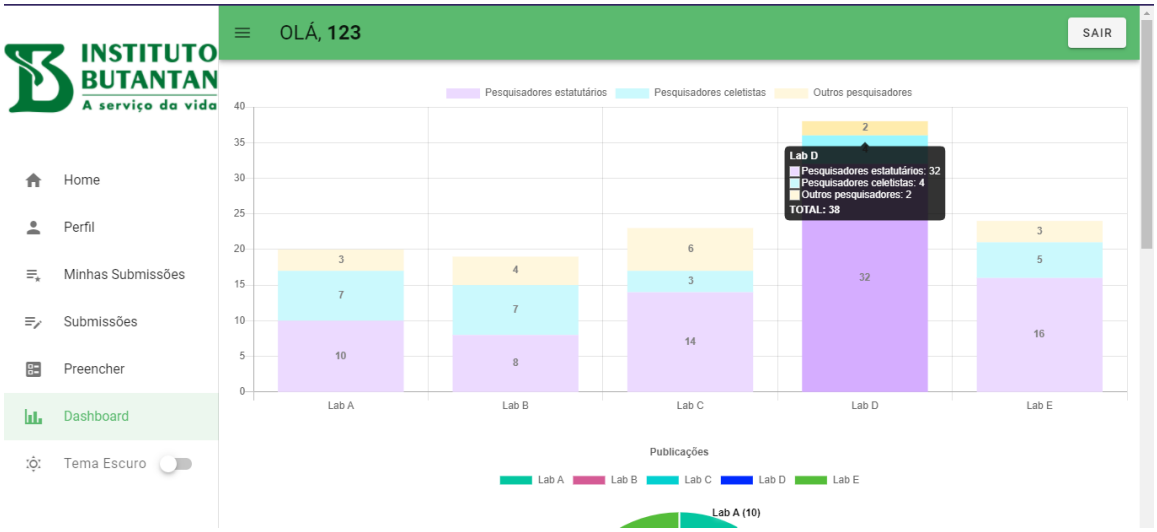


Figura 5.1: *Protótipo de gráfico de barras no sistema Scientiometer. Com dados fictícios de número de pesquisadores.*

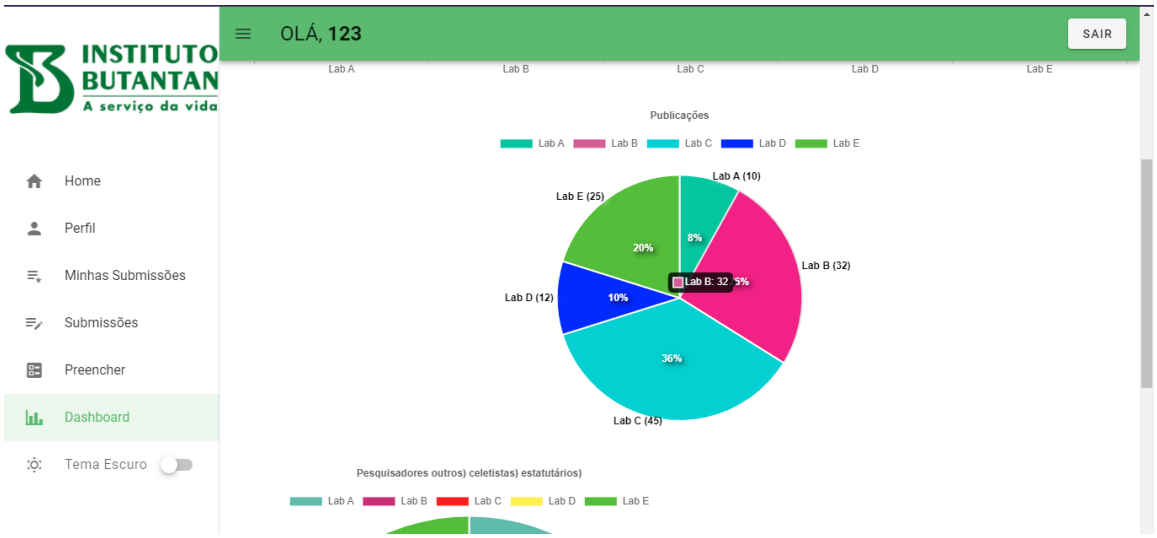


Figura 5.2: *Protótipo de gráfico de pizza no sistema Scientiometer. Com dados fictícios de número de pesquisadores.*

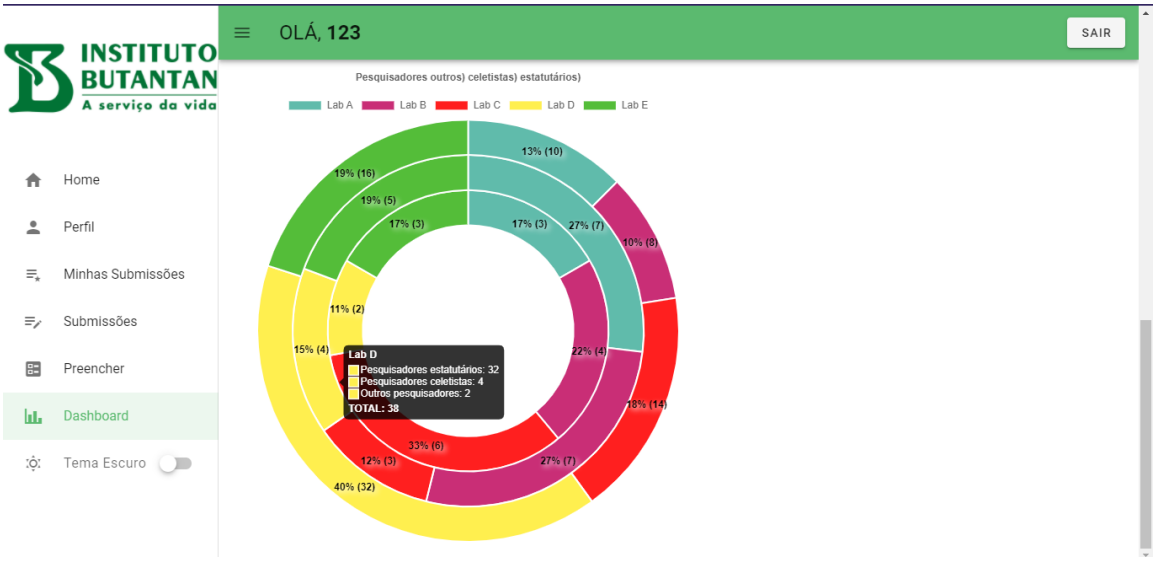


Figura 5.3: *Protótipo de gráfico de camadas no sistema Scientiometer. Com dados fictícios de número de pesquisadores.*

Referências

[Bornmann e Daniel 2005] Lutz Bornmann e Hans-Dieter Daniel. "Does the h-index for ranking of scientists really work?" Em: *Scientometrics* 65.3 (2005), pgs. 391–392 (citado na pg. 6).

[CAPES 2020] CAPES. *Qualis-Periódicos*. Acessado em 2 de dezembro de 2020. 2020. URL: https://sucupira.capes.gov.br/sucupira/public/index.xhtml (citado na pg. 6).

[Cleaner 2020] Database Cleaner. *Database Cleaner*. Versão 1.8.5. 2020. URL: https://github.com/DatabaseCleaner/database_cleaner (citado na pg. 18).

[CNPq 2020] CNPq. *Bolsas e Auxílios*. Acessado em 2 de dezembro de 2020. 2020. URL: http://www.cnpq.br/web/guest/apresentacao-bolsas-e-auxilios (citado na pg. 6).

[Faker-Ruby 2020] Faker-Ruby. *Faker*. Versão 2.15.1. 2020. URL: https://github.com/faker-ruby/faker (citado na pg. 18).

[Filipova 2016] Olga Filipova. *Learning Vue. js 2*. Packt Publishing Ltd, 2016 (citado na pg. 2).

[Fowler 2018] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018 (citado na pg. 7).

[Garfield 1999] Eugene Garfield. *Journal impact factor: a brief review*. 1999 (citado na pg. 6).

[Gupta 1979] BM Gupta. "Citation analysis and its applications: a review". Em: (1979) (citado na pg. 5).

[Huang e Kusiak 1996] Chun-Che Huang e Andrew Kusiak. "Overview of kanban systems". Em: (1996) (citado na pg. 19).

[James Lindenbaum 2020] Orion Henry James Lindenbaum Adam Wiggins. *Heroku cloud platform as a service (PaaS)*. Último acesso em 18 de dezembro de 2020. 2020. URL: https://www.heroku.com (citado na pg. 17).

[Krasner, Pope *et al.* 1988] Glenn E Krasner, Stephen T Pope *et al.* "A description of the model-view-controller user interface paradigm in the smalltalk-80 system". Em: *Journal of object oriented programming* 1.3 (1988), pgs. 26–49 (citado na pg. 18).

[Martins 2020] Paulo Egydio Martins. *Lei Complementar Nº 125, de 18 de novembro de 1975*. Acessado em 2 de dezembro de 2020. 2020. url: http://www.pesquisador.sp.gov.br/lei.html (citado na pg. 7).

[Mathias Biilmann 2020] Christian Bach Mathias Biilmann. *Netlify*. Último acesso em 22 de dezembro de 2020. 2020. url: https://www.netlify.com (citado na pg. 17).

[Momjian 2001] Bruce Momjian. *PostgreSQL: introduction and concepts*. Vol. 192. Addison-Wesley New York, 2001 (citado na pg. 1).

[Robert McCool 2020] Apache Software Foundation Robert McCool. *Apache Apache HTTP Server*. Último acesso em 29 de janeiro de 2021. 2020. url: https://httpd.apache.org/ (citado na pg. 32).

[Rspec 2013] Rspec. *Rspec-Rails*. Versão 4.0.1. 2013. url: https://github.com/rspec/rspec-rails (citado na pg. 18).

[Scholl 2019] Bruno B. Scholl. *Desenvolvimento de um sistema de informação para coleta e análise de relatórios de produtividade científica*. 2019 (citado nas pgs. 1, 5, 15).

[Scitable 2020] Scitable. *Scientific Papers*. Acessado em 30 de novembro de 2020. 2020. url: https://www.nature.com/scitable/topicpage/scientific-papers-13815490/ (citado na pg. 5).

[Silva e Bianchi 2001] José Aparecido Silva e Maria de Lourdes Pires Bianchi. "Cientometria: a métrica da ciência". Em: *Paidéia* 11.20 (2001), pgs. 5–10 (citado na pg. 5).

[Simplecov-Ruby 2020] Simplecov-Ruby. *SimpleCov*. Versão 0.18.0. 2020. url: https://github.com/simplecov-ruby/simplecov (citado na pg. 28).

[Smith 1981] Linda C Smith. "Citation analysis". Em: (1981) (citado na pg. 5).

[Taylor 2010] Barry Taylor. *Rails: a guide to rails, crakes, gallinules and coots of the world*. Bloomsbury Publishing, 2010 (citado na pg. 2).

[Thoughtbot 2020] Thoughtbot. *Factory Bot Rails*. Versão 6.1.0. 2020. url: https://github.com/thoughtbot/factory_bot_rails (citado na pg. 18).

[Tilkov e Vinoski 2010] Stefan Tilkov e Steve Vinoski. "Node. js: using javascript to build high-performance network programs". Em: *IEEE Internet Computing* 14.6 (2010), pgs. 80–83 (citado na pg. 17).

[Tom Preston-Werner 2020] P. J. Hyett Tom Preston-Werner Chris Wanstrath. *GitHub*. Último acesso em 22 de dezembro de 2020. 2020. url: https://github.com (citado na pg. 17).

[Torvalds 2020] Linus Torvalds. *Git*. Último acesso em 22 de dezembro de 2020. 2020. URL: https://git-scm.com (citado na pg. 17).

[Zhu *et al.* 1997] Hong Zhu, Patrick AV Hall e John HR May. "Software unit test coverage and adequacy". Em: *Acm computing surveys (csur)* 29.4 (1997), pgs. 366–427 (citado na pg. 27).