

**CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS  
CAMPUS TIMÓTEO**

Engenharia de Computação

Engenharia de Software

Trabalho prático de Teste de Software

André Marcelino de Souza Neves

[andreneves3@gmail.com](mailto:andreneves3@gmail.com)

Leonam Teixeira de Vasconcelos

[leonam.teixeira.vasconcelos@gmail.com](mailto:leonam.teixeira.vasconcelos@gmail.com)

Professora: Deisymar Botega Tavares

**Timóteo  
2019**

# Sumário

<b>Lista de ilustrações</b> . . . . .	<b>3</b>	
<b>Lista de tabelas</b> . . . . .	<b>4</b>	
<b>1</b>	<b>INTRODUÇÃO</b> . . . . .	<b>5</b>
1.1	O sistema . . . . .	5
<b>2</b>	<b>REVISÃO BIBLIOGRÁFICA</b> . . . . .	<b>6</b>
2.1	Testes de unidade . . . . .	6
2.2	Testes de cobertura . . . . .	6
2.3	Testes de sistema . . . . .	7
2.4	Jest . . . . .	7
<b>3</b>	<b>DESENVOLVIMENTO</b> . . . . .	<b>8</b>
3.1	<b>Teste Unitário e de Cobertura</b> . . . . .	<b>8</b>
3.1.1	Código . . . . .	9
3.1.2	Complexidade ciclomática . . . . .	13
3.1.3	Casos de Testes Unitário . . . . .	14
3.1.4	Automação dos Testes . . . . .	15
3.1.4.1	Ferramenta utilizada . . . . .	15
3.1.4.2	Implementação dos testes . . . . .	15
3.1.4.3	Resultados obtidos . . . . .	15
3.2	<b>Teste de Sistema</b> . . . . .	<b>18</b>
3.3	<b>Caso de uso 1: Cadastro com mais dados: Tópicos de curso</b> . . . . .	<b>19</b>
3.3.1	Descrição . . . . .	19
3.3.2	Print Screen das telas . . . . .	19
3.3.3	Procedimento de teste . . . . .	22
3.3.4	Caso de teste . . . . .	23
3.4	<b>Caso de uso 2: Submissão de Exercício</b> . . . . .	<b>25</b>
3.4.1	Fluxo de Submissão e Print Screen das telas . . . . .	26
3.4.2	Procedimento de teste . . . . .	31
<b>4</b>	<b>CONCLUSÃO</b> . . . . .	<b>33</b>
<b>REFERÊNCIAS</b> . . . . .	<b>34</b>	

# Lista de ilustrações

Figura 1 – Tela inicial do sistema PediDoctor. . . . .	5
Figura 2 – Código da função testada [Parte 1]. . . . .	10
Figura 3 – Código da função testada [Parte 2]. . . . .	11
Figura 4 – Vetor de dados [Parte 1]. . . . .	11
Figura 5 – Vetor de dados [Parte 2]. . . . .	12
Figura 6 – Grafo de complexidade ciclomática do caso de teste. . . . .	13
Figura 7 – Diagrama de regiões no grafo de fluxo do caso de teste. . . . .	14
Figura 8 – Implementação da automação dos testes unitário e de cobertura. . . . .	16
Figura 9 – Terminal com o resultado das segunda execução dos testes unitários e de cobertura. . . . .	17
Figura 10 – Exibição da ramificação não coberta pelo primeiro conjunto de testes unitários. . . . .	17
Figura 11 – Terminal com o resultado das segunda execução dos testes unitários e de cobertura. Agora, verifica-se que não houve nenhum erro, e obteve completa cobertura. . . . .	18
Figura 12 – Tela de Gerenciar tópicos de curso no qual também faz uma listagem de todos os tópicos. . . . .	20
Figura 13 – Tela de cadastrar tópicos. . . . .	20
Figura 14 – Começo da tela de cadastrar tópicos mostrando os campos que são obrigatórios. . . . .	20
Figura 15 – Fim da tela de cadastrar tópicos mostrando os campos que são obrigatórios. . . . .	21
Figura 16 – Começo da tela de editar tópicos. . . . .	21
Figura 17 – Fim da tela de cadastrar tópicos. . . . .	22
Figura 18 – Tela de excluir um tópico. . . . .	22
Figura 19 – Fotografias da tela de submissão. . . . .	25
Figura 20 – Tela inicial(Painel) do módulo aluno. . . . .	26
Figura 21 – Tela de listagem de aluno . . . . .	27
Figura 22 – Seleção das linguagens . . . . .	27
Figura 23 – Conteúdo do "Dropdown" . . . . .	28
Figura 24 – Selecionando um arquivo . . . . .	28
Figura 25 – Navegação interna . . . . .	29
Figura 26 – Navegação interna . . . . .	29
Figura 27 – Confirmação da submissão . . . . .	30
Figura 28 – Confirmação da submissão . . . . .	30

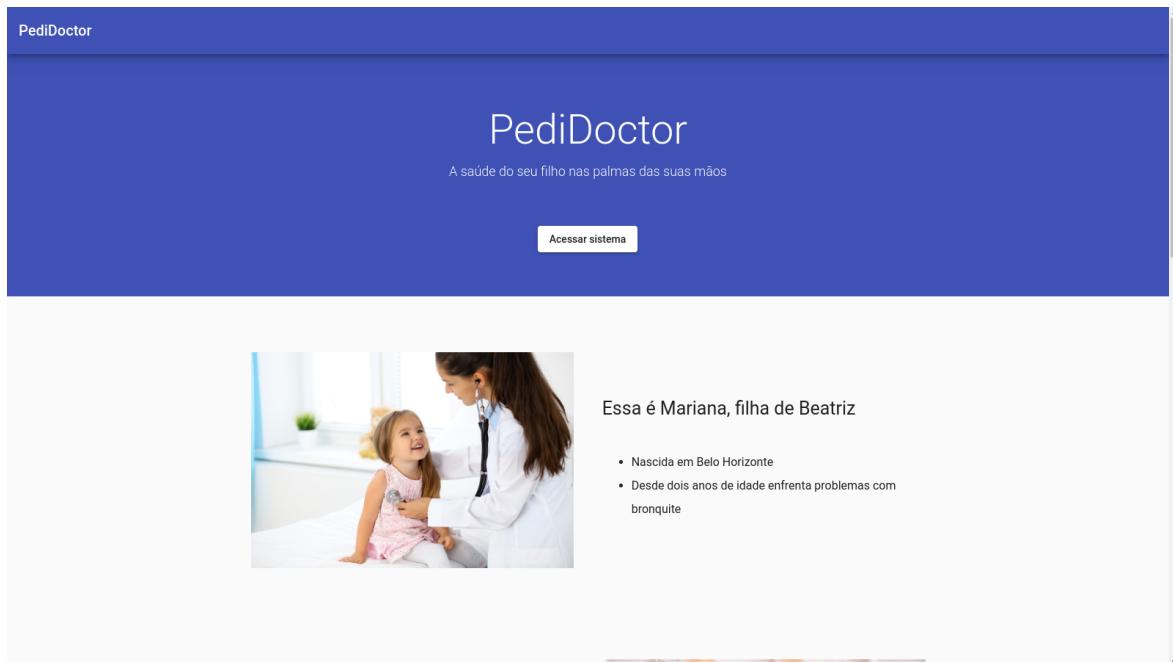
# Lista de tabelas

Tabela 1 – Itens resumidos para referência de cálculo de dosagem . . . . .	8
Tabela 2 – Casos de teste unitário . . . . .	14
Tabela 4 – Caso de teste de sistema - Tópicos de curso . . . . .	24
Tabela 5 – Caso de teste de sistema - Submissão de exercício . . . . .	31

# 1 Introdução

Este documento apresenta o relato de testes realizado no sistema **PediDoctor**. O sistema foi desenvolvido pelos alunos **André Marcelino de Souza Neves** e **Leonam Teixeira de Vasconcelos**, do curso de **Engenharia de Computação**, da instituição **CEFET MG - Campus Timóteo**, na disciplina **Engenharia de Software**.

Figura 1 – Tela inicial do sistema PediDoctor.



Fonte: autores.

O sistema foi testado por **Adriel Vinícius Moraes Araújo** e **Maycon Arthuso de Carvalho**, também alunos da disciplina.

## 1.1 O sistema

O projeto tem como objetivo desenvolver um sistema voltado para atendimento médico de pediatria. O propósito é resolver problemas que existem quando um os pais trocam a clínica na qual realiza consultas de seus filhos. Nessa situação, todo o histórico médico do paciente é perdido, o que causa problemas para os próximos médicos avaliarem a saúde da criança com base em seu histórico.

Para isso, **PediDoctor** tem como objetivo ser uma plataforma unificada, que mantém, de forma estruturada, as consultas e diagnósticos obtidos ao longo do tempo, com detalhamento de informações como sintomas sentidos, doenças diagnosticadas e medicamentos receitados. Isso permite que, ao longo do tempo, o histórico seja mantido, ainda que o paciente torne-se cliente de outras clínicas.

## 2 Revisão bibliográfica

Nessa seção, alguns termos utilizados nesse documento serão esclarecidos, com uma breve e referenciada revisão bibliográfica dos assuntos aqui tratados. Será descrito sucintamente as características dos tipos de testes abordados, e as ferramentas utilizadas para tal.

### 2.1 Testes de unidade

Segundo Sommerville (SOMMERVILLE, 2011), um teste de unidade – referido pelo autor como teste unitário – é o processo de testar os componentes de programa: métodos, classes etc. Deve-se testar:

- Todas as operações daquele objeto;
- Verificar os valores de todos os atributos daquele objeto;
- Analisar todos os estados possíveis do objeto.

### 2.2 Testes de cobertura

O teste de cobertura tem por objetivo verificar se os testes realizados passaram por vários fluxos diferentes da aplicação, analisando justamente quais partes do código foram cobertas ou não por um conjunto de casos de testes (Devmedia, 2011).

Esse teste é importante para verificarmos se um teste é completo o suficiente para que todas as possibilidades de estado de um componente ou função sejam alcançadas, e quando o mesmo seja colocado em produção, um caso não testado não cause maiores problemas.

Outro motivo para utilizar testes de cobertura é a análise de código para verificarmos a inutilidade de determinados trechos. Se temos um conjunto completo de casos de testes e determinado trecho de código nunca é alcançado, podemos concluir que aquele código não possui muita utilidade no software como um todo. Obviamente, deve-se tomar cuidado com as decisões tomadas nesse contexto para que não haja a remoção de partes preciosas de um software baseando-se em uma análise superficial.

```

1: function teste_idade ( idade ) {
2:     if ( idade < 18 )
3:         return false
4:     else
5:         return true
6:     return false
7: }
```

No código acima, podemos verificar um exemplo extremamente simples, mas que pode ser utilizado para o entendimento do conceito apresentado. Por mais complexo e avançado que seja um conjunto de casos de testes desenvolvido para testar a função acima, nenhum deles alcançará a linha 6, mostrando no entanto, a inutilidade dessa linha de código.

## 2.3 Testes de sistema

De acordo com Pressman (PRESSMAN, ), um teste de sistema consiste em não apenas um grupo de testes semelhantes, mas sim e um conjunto de variados tipos de testes com o objetivo de exercitar um sistema como um todo. Entre os testes comumente utilizados podemos citar:

- Recuperação;
- Segurança;
- Desempenho;
- Disponibilização;
- entre outros.

## 2.4 Jest

Jest é um framework de testes de código aberto para a linguagem javascript com foco em simplicidade e minimalismo. Todo o código do software pode ser encontrado em seu repositório oficial no *github*<sup>1</sup> e informações sobre uso e instruções no site oficial <sup>2</sup>.

O Jest foi projetado para funcionar sem a necessidade de demasiada configuração, algo que outros frameworks de testes não alcançaram até os dias de hoje. Softwares como Karma-Jasmine que é uma outra alternativa para testes e vem por padrão nos projetos Angular (framework utilizado para programar a interface gráfica do PediDoctor) demandam configuração pesada e extensa para se realizar os testes necessários.

Para testar com o jest, criamos arquivos de teste para o objeto que desejamos testar e executamos o teste através de um terminal, com o comando jest. Para o teste de cobertura, apenas adicionamos a flag -coverage no comando anterior, ficando assim: jest -coverage (Os comandos devem ser executados, obviamente após termos adicionado e instalado as dependências do Jest no projeto).

---

<sup>1</sup> <<https://github.com/facebook/jest>>

<sup>2</sup> <<https://jestjs.io/en/>>

### 3 Desenvolvimento

O desenvolvimento desse trabalho foi dividido em duas partes: A primeira teve o objetivo de estudar e utilizar uma ferramenta para automação de testes unitários e de cobertura. A ferramenta utilizada foi o *Framework Jest*, citado na seção 2.4.

A segunda parte teve como objetivo testar o software da outra dupla de forma trocada.

#### 3.1 Teste Unitário e de Cobertura

Para essa parte, foi escolhida a função **Cálculo de dosagem de medicamentos**. Em contextos reais, existem tabelas que indicam a dosagem adequada para cada medicamento, de acordo com o peso e idade da criança. Para esse momento, foi considerada a tabela **Sugestões de Doses em Pediatria**, elaborada por *Jean Roberge (Pediatra HRO)* e *Camila Biazi (Doutoranda UNOCHAPECÓ)*<sup>1</sup>

Para simplificação da implementação, foi considerado um subconjunto do material de referência, mostrado na Tabela 1.

Tabela 1 – Itens resumidos para referência de cálculo de dosagem

Cod.	Nome do Medicamento	Dosagem
01	DIPIRONA Gotas 500mg/ml	Dose: 1 gota/kg. Int.: 6/6 horas. (Dose máx.: 40 gotas)
02	DIPIRONA Injetável	Dose (ml): Peso / 20. (Dose máx.: 2 ml)
03	CARBOCISTEÍNA Xarope Infantil 20 mg/ml	<ul style="list-style-type: none"> <li>• De 1 a 4 anos: 2,5 ml</li> <li>• De 5 a 10 anos: 5 ml</li> <li>• Acima de 10 anos: 10 ml</li> </ul> <p>Intervalo: 8/8 horas</p>

<sup>1</sup> <<https://pdfslide.net/documents/macetes-pediatria.html>>

04	RANITIDINA Solução Oral	<ul style="list-style-type: none"><li>• 4 Kg ou abaixo: Dose: 0,125 ml / kg.</li><li>• 5 a 7 Kg: Dose: 1 ml.</li><li>• 8 a 10 Kg: Dose: 1,5 ml.</li><li>• Acima de 11 Kg: Dose: 2 ml.</li></ul> <p>Intervalo: 12/12 horas</p>
----	-------------------------	---

Fonte: Jean Roberge (Pediatra HRO) e Camila Biazi (Doutoranda UNOCHAPECÓ)

### 3.1.1 Código

O código da função desenvolvida para está mostrado abaixo em Figura 2 e Figura 3. O código considera os intervalos de idade / peso especificados na tabela, além de considerar casos de dosagem máxima.

O vetor de dados utilizado que definiram o estado do sistema (banco de dados) está mostrado em Figura 4 e Figura 5

Figura 2 – Código da função testada [Parte 1].

```

calcudoSagamento(cod : number, idade : number, peso : number) {

    const med = this.getMedicamento(cod); 1
    let comparacaoFaixas, dose;

    switch(med.posologia.classificacao) { 2
        case 'peso': 3
            comparacaoFaixas = peso;
            break; 4

        case 'idade': 5
            comparacaoFaixas = idade;
            break; 6

    }

    let faixa = null; 7

    for(let i = 0; i < med.posologia.faixas.length; i++) { 8
        const atual = med.posologia.faixas[i]; 9
        10

        if( 11
            atual.min == 0 && atual.max == 0 12
            atual.min == 0 && comparacaoFaixas <= atual.max 13
            atual.max == 0 && comparacaoFaixas >= atual.min 14
            comparacaoFaixas <= atual.max && comparacaoFaixas >= atual.min 15
        ) { 16

            faixa = atual; 17
            break; 18

        } 19
    } 20
}

```

Figura 3 – Código da função testada [Parte 2].

```

dose = faixa.valorFixo + faixa.proporcional * comparacaoFaixas; 21
dose = Math.floor(dose * 100) / 100;

if(med.posologia.maxima > 0) 22
    dose = Math.min(dose, med.posologia.maxima); 23

dose = `${dose}`.replace('.', ','); 24

const intervalo = med.posologia.intervalo; 25
let intervaloMsg = '';

if(intervalo > 0) {26
    intervaloMsg = ` de ${ intervalo } em ${ intervalo } horas` 27
}

return `Posologia: ${ dose } ${ med.posologia.tipoDose }${intervaloMsg}.`;
} 28

```

Figura 4 – Vetor de dados [Parte 1].

```

medicamentos = [
{
    cod: 1, nome: 'DIPIRONA Gotas 500mg/ml',
    posologia: {
        intervalo: 6, tipoDose: 'gotas', maxima: 40, classificacao: 'peso',
        faixas: [
            { min: 0, max: 0, valorFixo: 0, proporcional: 1 },
        ]
    }
},
{
    cod: 2, nome: 'DIPIRONA Injetável',
    posologia: {
        intervalo: 0, tipoDose: 'ml', maxima: 2, classificacao: 'peso',
        faixas: [
            {min: 0, max: 0, valorFixo: 0, proporcional: 1 / 20},
        ]
    }
},

```

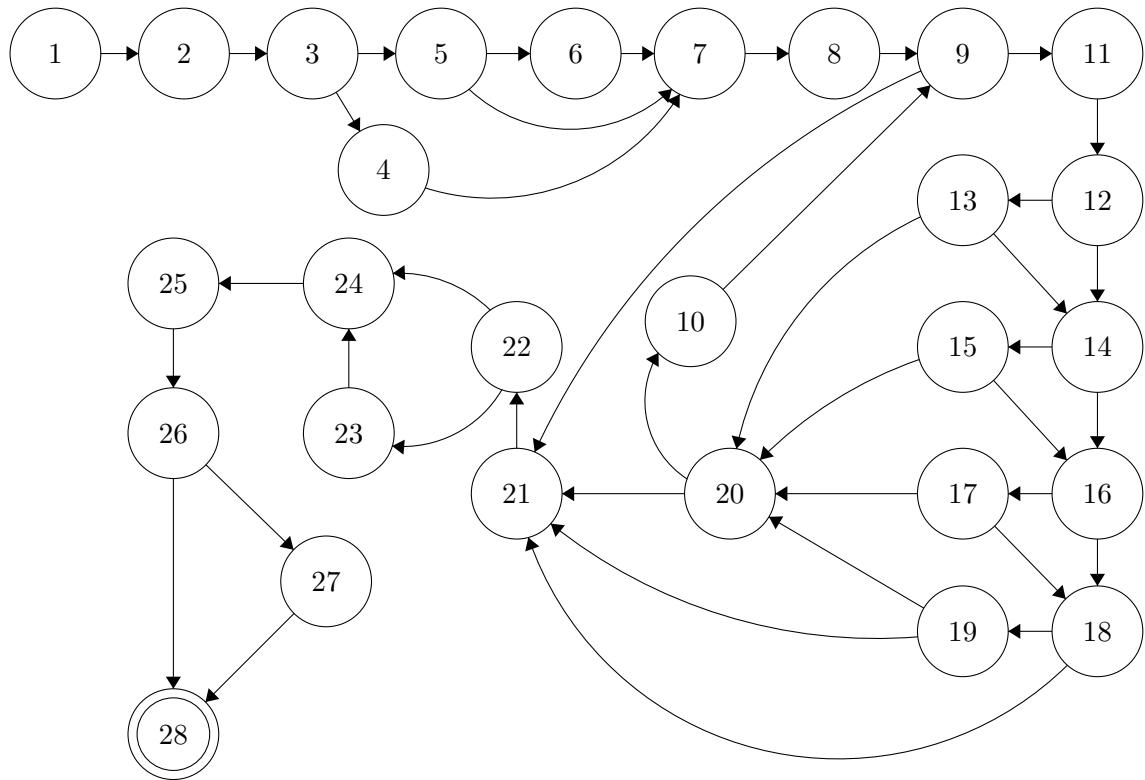
Figura 5 – Vetor de dados [Parte 2].

```
cod: 3, nome: 'CARBOCISTEÍNA Xarope Infantil 20 mg/ml',
  posologia: {
    intervalo: 8,
    tipoDose: 'ml',
    maxima: 0,
    classificacao: 'idade',
    faixas: [
      {min: 1, max: 4, valorFixo: 2.5, proporcional: 0},
      {min: 5, max: 10, valorFixo: 5, proporcional: 0},
      {min: 10, max: 0, valorFixo: 10, proporcional: 0},
    ]
  }
},
{
  cod: 4, nome: 'RANITIDINA Solução Oral',
  posologia: {
    intervalo: 12, tipoDose: 'ml', maxima: 0, classificacao: 'peso',
    faixas: [
      {min: 0, max: 4, valorFixo: 0, proporcional: 0.125},
      {min: 5, max: 7, valorFixo: 1, proporcional: 0 },
      {min: 8, max: 10, valorFixo: 1.5, proporcional: 0 },
      {min: 11, max: 0, valorFixo: 2, proporcional: 0},
    ]
  }
};
];
```

### 3.1.2 Complexidade ciclomática

A partir do código desenvolvido, foi obtido o grafo da complexidade ciclomática, mostrado na Figura 6.

Figura 6 – Grafo de complexidade ciclomática do caso de teste.



**Fonte:** autores.

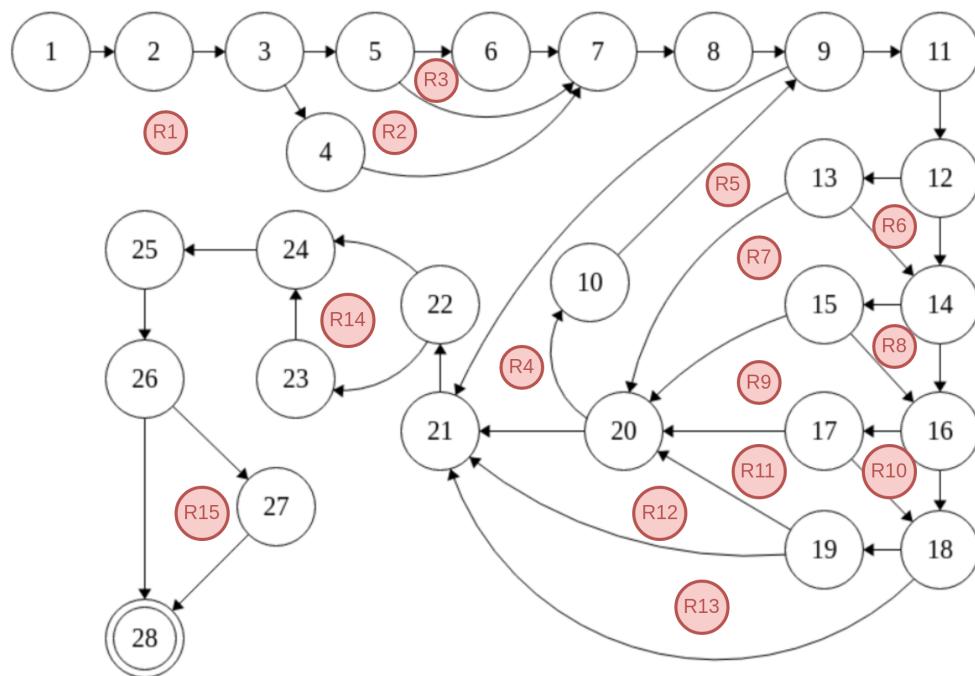
O número da complexidade ciclomática do fluxo da função representa o número de caminhos independentes para percorrer a função. Ele pode ser calculado com base no número de nós predicados, isso é, o número de nós que possuem no mínimo duas arestas que partem dele. O valor alcançado para esse grafo é  $V(G) = 15$ . Os caminhos independentes são:

1. 1-2-3-4-7-8-9-11-12-13-14-15-20-10-9-...
2. 1-2-3-4-7-8-9-11-12-13-14-15-20-21-22-23-24-25-26-27-28
3. 1-2-3-4-7-8-9-11-12-13-14-15-20-21-22-23-24-25-26-28
4. 1-2-3-4-7-8-9-11-12-13-14-15-20-21-22-24-25-26-27-28
5. 1-2-3-4-7-8-9-11-12-13-14-15-20-21-22-24-25-26-28
6. 1-2-3-4-7-8-9-11-12-13-14-15-16-17-20-21-22-24-25-26-28
7. 1-2-3-4-7-8-9-11-12-13-14-16-18-19-21-22-24-25-26-28

8. 1-2-3-4-7-8-9-11-12-13-20-21-22-24-25-26-28
9. 1-2-3-4-7-8-9-11-12-14-15-20-21-22-24-25-26-28
10. 1-2-3-4-7-8-9-21-22-23-24-25-26-27-28
11. 1-2-3-4-7-8-9-21-22-23-24-25-26-28
12. 1-2-3-4-7-8-9-21-22-24-25-26-27-28
13. 1-2-3-4-7-8-9-21-22-24-25-26-28
14. 1-2-3-5-6-7-8-9-11-12-13-20-21-22-24-25-26-28
15. 1-2-3-5-7-8-9-11-12-13-20-21-22-24-26-28

A figura abaixo ilustra o número de regiões possíveis no grafo de fluxo da funcionalidade, que também é igual ao valor da complexidade ciclomática do sistema:

Figura 7 – Diagrama de regiões no grafo de fluxo do caso de teste.



Fonte: autores.

### 3.1.3 Casos de Testes Unitário

A Tabela 2 mostra o registro dos testes realizados. De acordo com os padrões da função, cada faixa de idade / peso representa uma classe de equivalência. Todos os casos de testes são baseados na técnica de *Análise do Valor Limite*.

Tabela 2 – Casos de teste unitário

1. Projeto	PediDoctor					
2. Testadores	André e Leonam					
3. Identificação	PEDI-CTU-01					
4. Data da elaboração	11/11/2019					
5. Itens a testar	Função <i>Cálculo de dosagem</i> , que recebe como parâmetros: Identificação do medicamento, peso e idade da criança.					
6. Entradas	Campo	Valores				
		T1	T2	T3	T4	T5
	Cod.	1	2	3	4	4
	Idade	4	4	11	0	0
	Peso	41	39	30	4	10
7. Saídas esperadas	T1	Posologia: 40 gotas de 6 em 6 horas.				
	T2	Posologia: 1,95 ml.				
	T3	Posologia: 10 ml de 8 em 8 horas.				
	T4	Posologia: 0,5 ml de 12 em 12 horas.				
	T5	Posologia: 1,5 ml de 12 em 12 horas.				
8. Resultado da aplicação do caso de teste	T1	Teste aprovado				
	T2	Teste aprovado				
	T3	Teste aprovado				
	T4	Teste aprovado				
	T5	Teste aprovado				
Legenda	Verde: Teste executado e nenhum erro encontrado.					
	Vermelho: teste executado, erro encontrado e aguardando correção.					
	Obs: legenda para ser utilizada sobre o texto do resultado o item 8					

Fonte: autores.

### 3.1.4 Automação dos Testes

#### 3.1.4.1 Ferramenta utilizada

Para automação dos testes unitário e de cobertura, foi utilizada o *Framework* de testes *Jest*, específico para códigos em javascript, conforme já citado na seção 2.4. A versão utilizada foi 24.9.

#### 3.1.4.2 Implementação dos testes

A Figura 8 exibe o código utilizado na implementação da automação dos testes.

#### 3.1.4.3 Resultados obtidos

Foram feitas duas execuções: Na primeira, havia apenas os casos de teste de **T1** a **T4**.

Em ambas, houve total aceitação dos resultados dos testes. No entanto, foi verificado que os não havia total cobertura. A Figura 9 exibe o resumo da execução, e a Figura 10 mostra a visualização gerada pela ferramenta, que evidencia uma parte do código não executada pelos testes.

Figura 8 – Implementação da automação dos testes unitário e de cobertura.



```

import { Test, TestingModule } from '@nestjs/testing';
import { MedicamentoDosagemService } from './medicamento-dosagem.service';

describe('MedicamentoDosagemService', () => {
  const service = new MedicamentoDosagemService();

  it('Teste 1', () => {
    expect(
      service.calculoDosagem(1, 4, 41)
    ).toBe('Posologia: 40 gotas de 6 em 6 horas.');
  });

  it('Teste 2', () => {
    expect(
      service.calculoDosagem(2, 4, 39)
    ).toBe('Posologia: 1,95 ml.');
  });

  it('Teste 3', () => {
    expect(
      service.calculoDosagem(3, 11, 30)
    ).toBe('Posologia: 10 ml de 8 em 8 horas.');
  });

  it('Teste 4', () => {
    expect(
      service.calculoDosagem(4, 0, 4)
    ).toBe('Posologia: 0,5 ml de 12 em 12 horas.');
  });

  it('Teste 5', () => {
    expect(
      service.calculoDosagem(4, 0, 10)
    ).toBe('Posologia: 1,5 ml de 12 em 12 horas.');
  });
});

```

Essas sequências condicionais foram utilizadas para verificar em qual faixa de dosagem a entrada se encaixa. Como é possível ver na Tabela 1, existem casos, como os medicamentos **03** e **04**, em que a dosagem varia conforme a faixa de peso e idade. Na Figura 4 e Figura 5, é possível ver a definição dessas faixas (dentro do vetor *faixas[]*). Quando um dos parâmetros *min* ou *max* são iguais a zero, isso representa uma situação onde não há limite mínimo ou máximo, respectivamente. Há também os casos onde não há nem limite inferior e nem superior, mas sim uma faixa única, como é o caso dos medicamentos **01** e **02**.

Na primeira execução, foram incluídos apenas os testes de **T1** a **T4**, conforme descrito na Tabela 2. Em relação ao trecho do código com falha de cobertura, na Figura 10, os testes **T1** e **T2**

Figura 9 – Terminal com o resultado das segunda execução dos testes unitários e de cobertura.

```

PASS  src/domain/pedilandia/medicamento-dosagem/medicamento-dosagem.service.spec.ts (9.958s)
  MedicamentoDosagemService
    ✓ Teste 1 (3ms)
    ✓ Teste 2
    ✓ Teste 3
    ✓ Teste 4 (1ms)

-----| % Stmt | % Branch | % Funcs | % Lines | Uncovered Line #s |
-----| 100   | 93.75  | 100     | 100     | 100     | 179
All files
  medicamento-dosagem.service.ts | 100   | 93.75  | 100     | 100     | 100     | 179

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        10.163s
Ran all test suites.

Finish! Open http://localhost:8080/lcov-report/ to see the report

```

Figura 10 – Exibição da ramificação não coberta pelo primeiro conjunto de testes unitários.

```

if(

  atual.min == 0 && atual.max == 0           ||
  atual.min == 0 && comparacaoFaixas <= atual.max ||
  atual.max == 0 && comparacaoFaixas >= atual.min ||
  comparacaoFaixas <= atual.max && comparacaoFaixas >= atual.min

) {

  faixa = atual;
  break;

}

```

abrangeram a primeira condição ( $min == 0$  e  $max == 0$ ). O teste **T3** abrangeu a terceira condição ( $max == 0$ ), pois a faixa específica compreende crianças com idade a partir de dez anos. Por fim, o teste **T4** abrangeu a segunda condição ( $min == 0$ ), pois a faixa específica compreende crianças com peso menor ou igual a quatro.

Contudo, nenhum teste abrangeu a quarta condição, isto é, uma faixa de dosagem com limites máximo e mínimos bem definidos. Com isso, foi acrescentado o teste **T5**, o qual enquadra-se na faixa entre 8 Kg e 10 Kg, e torna necessária a verificação que antes não foi coberta pelos testes. O resultado dos testes após acrescentar **T5** está mostrado na Figura 11.

Figura 11 – Terminal com o resultado das segunda execução dos testes unitários e de cobertura. Agora, verifica-se que não houve nenhum erro, e obteve completa cobertura.

```

PASS  src/domain/pedilandia/medicamento-dosagem/medicamento-dosagem.service.spec.ts
MedicamentoDosagemService
  ✓ Teste 1 (2ms)
  ✓ Teste 2
  ✓ Teste 3
  ✓ Teste 4 (1ms)
  ✓ Teste 5

-----|-----|-----|-----|-----|-----|
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
-----|-----|-----|-----|-----|-----|
All files | 100 | 100 | 100 | 100 | |
medicamento-dosagem.service.ts | 100 | 100 | 100 | 100 | |
-----|-----|-----|-----|-----|-----|
Test Suites: 1 passed, 1 total
Tests:      5 passed, 5 total
Snapshots:   0 total
Time:        1.279s
Ran all test suites.

Finish! Open http://localhost:8080/lcov-report/ to see the report

```

### 3.2 Teste de Sistema

O conteúdo textual a seguir é referente ao teste realizado no Software Desenvolvido por *Adriel Vinícius* e *Maycon Arthuso*, ambos alunos da disciplina Engenharia de Software, ministrada pela professora *Deysimar Botega Tavares* no período segundo do ano de 2019. A descrição do projeto criado está a seguir.

O ambiente universitário é permeado por diversos desafios para os alunos e professores. Para os discentes, a dificuldade gerada pela inexperiência e a exigência de vencer as várias curvas de aprendizado dentro de um único semestre, tornam a computação e a lógica de programação motivadores para baixo rendimentos, frustrações e, consequentemente em alguns casos, desistências. Para docentes, a pressão por cumprimento de datas, a entrega constante de material científico, preparação de aulas, correções de provas e atividades de muitas turmas, causam problemas a saúde mental e física, sem contar a insatisfação dos alunos.

É neste contexto que surgiu a ideia por parte do professor Aléssio Jr. de construir uma plataforma online de apoio aos alunos e professores. Unindo conceitos de programação competitiva com as necessidades do ambiente atarefado e estressante da universidade, a ideia tomou forma através de uma plataforma conhecida como *Maratona*. Esta plataforma pode ser encontrada e explorada através do endereço <<https://maratona.alessiojr.com/public>>. Apesar dos esforços, esta plataforma não atende todas às expectativas e deixa diversos problemas não resolvidos, tornando-se insatisfatório para os alunos e não adaptado para professores.

Com o passar do tempo, a ideia foi sendo construída devagar até atingir o status de startup em desenvolvimento. Neste ponto, a necessidade de validação comercial tornou-se obrigatória e outro protótipo foi construído buscando as adaptações necessárias juntamente com uma interface amigável. Esse protótipo recebeu o nome atual de *Mundo do código* e também pode ser encontrado através do endereço <<https://www.mundocodigocom/>>.

O sucesso deste último protótipo gerou o desejo de tornar o módulo de aluno, contido nessa

plataforma, mais robusto e preparado para uso massivo. Para isso, foram escolhidas duas ferramentas para construção do sistema; Elas são: Spring boot (API Java) e Angular Js.

Portanto, é neste contexto que a proposta apresentada neste documento se insere.

Visando testar os mais significativos casos de uso do sistema construído, os dois casos de uso escolhidos para esse trabalho são:

1. Cadastro com mais dados: Tópicos de curso
2. Submissão de Exercício

### 3.3 Caso de uso 1: Cadastro com mais dados: Tópicos de curso

O primeiro caso de uso será o de cadastro de dados referente ao tópicos de um curso.

#### 3.3.1 Descrição

Uma das responsabilidades do professor é preparar o conteúdo de sua aula. Este conteúdo, pode ser baseado em livros, apostilas, artigos e etc. Um conteúdo bem estruturado e claro, tem uma grande importância para o aprendizado rápido e eficaz por parte do aluno.

Essa funcionalidade de tópico de curso, tem por objetivo explicar a teoria e conceitos usados na programação, desde o mais básico, passando por aprendizado de estruturas de repetição e seleção, até estruturas mais complexas como árvores e grafos, para que o aluno aprenda o conteúdo, e posteriormente possa resolver exercícios que contenham tópicos abordados neste curso.

Um curso, cadastrado pelo usuário professor, tem vários assuntos, aqui chamados de *Tópicos de Curso*. Um mesmo tópico de curso pode estar em vários cursos diferentes, dependendo somente da forma como o professor estrutura o curso. O curso pode ser cadastrado, editado, excluído e listado, caso o usuário professor queira fazer alguma alteração.

#### 3.3.2 Print Screen das telas

Os print screen das telas de Tópicos de Curso seguem nas figuras a seguir. Os prints screen vão desde as telas para o cadastro de um tópico até visualização, edição, e exclusão do tópico.

Figura 12 – Tela de Gerenciar tópicos de curso no qual também faz uma listagem de todos os tópicos.

ID	Nome	Descrição	Ordem	Imagem	Data Criação	Data Atualização	Curso	Detalhar	Editar	Deletar
1	Introdução		1	👤	22/11/2019 10:28:13		Programação básica			
2	Entrada e Saída		2	💻	31/10/2019 12:42:05		Programação básica			
3	Principais Erros		2	🛠	31/10/2019 12:42:05		Programação básica			
4	Seleção		3	🚶	31/10/2019 12:42:05		Programação básica			
5	Seleção (parte 2)		3	🚶	31/10/2019 12:42:05		Programação básica			
6	Organização de Código		3	💻	31/10/2019 12:42:05		Programação básica			

Fonte: autores.

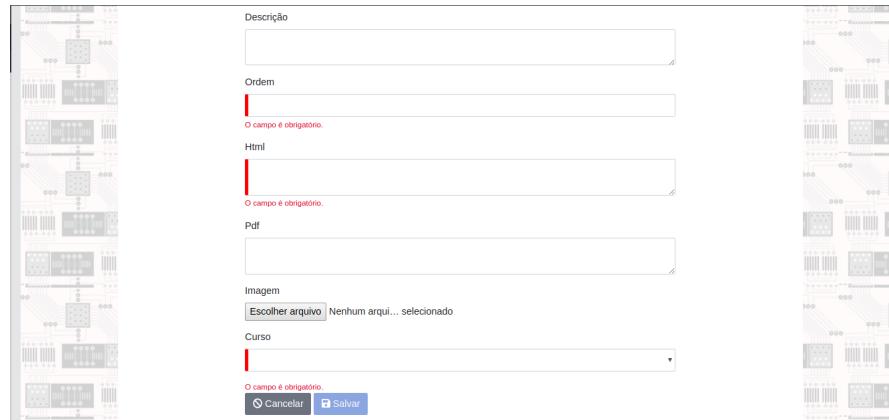
Figura 13 – Tela de cadastrar tópicos.

Fonte: autores.

Figura 14 – Começo da tela de cadastrar tópicos mostrando os campos que são obrigatórios.

**Fonte:** autores.

Figura 15 – Fim da tela de cadastrar tópicos mostrando os campos que são obrigatórios.



Descrição

Ordem

O campo é obrigatório.

Html

O campo é obrigatório.

Pdf

Imagem

Escolher arquivo Nenhum arquivo... selecionado

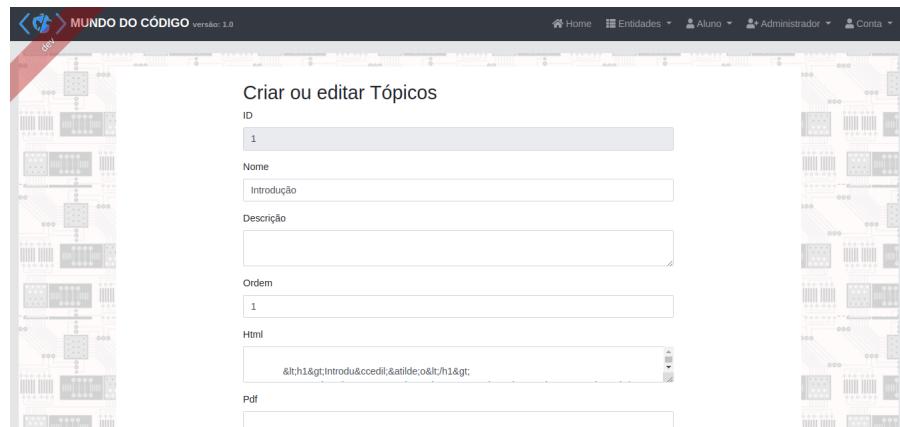
Curso

O campo é obrigatório.

Cancelar Salvar

**Fonte:** autores.

Figura 16 – Começo da tela de editar tópicos.



MUNDO DO CÓDIGO versão: 1.0

Home Entidades Aluno Administrador Conta

Criar ou editar Tópicos

ID

1

Nome

Introdução

Descrição

Ordem

1

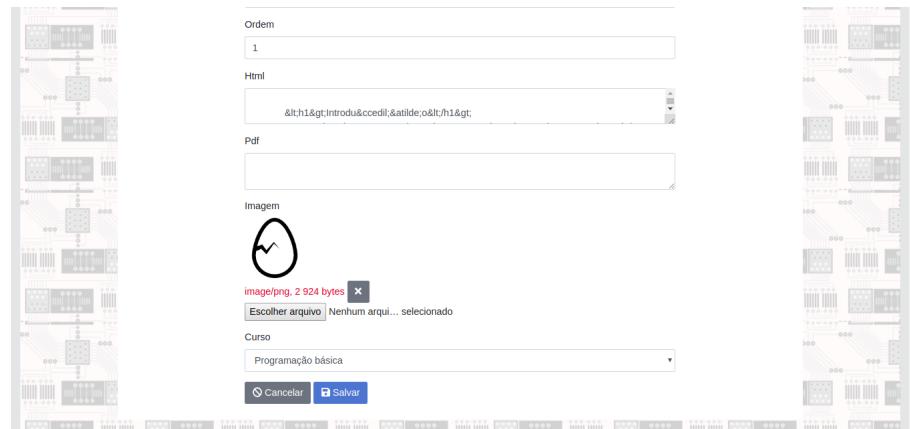
Html

&lt;h1&gt;Introdu&ccedil;&atilde;o&lt;/h1&gt;

Pdf

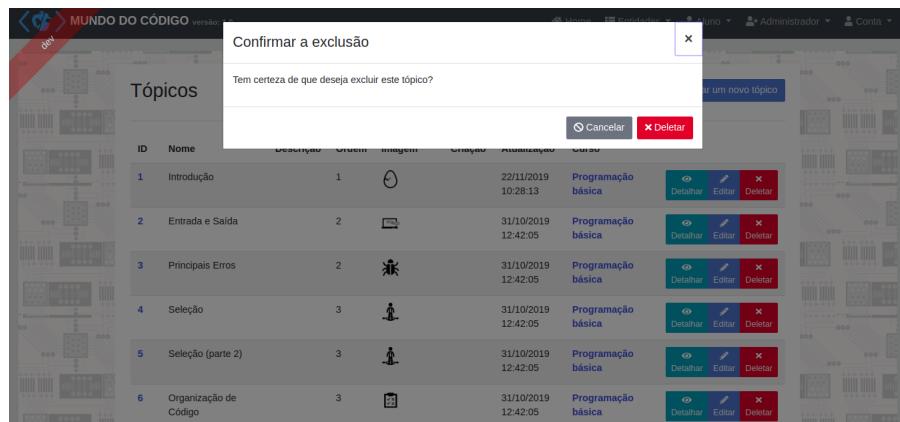
**Fonte:** autores.

Figura 17 – Fim da tela de cadastrar tópicos.



Fonte: autores.

Figura 18 – Tela de excluir um tópico.



Fonte: autores.

### 3.3.3 Procedimento de teste

- Objetivo: Verificar se o Tópico de Curso é feito corretamente
- Fluxo:
  1. Acessar painel principal
  2. Acionar comando "Criar um novo tópico"
  3. Inserir todos os campos
  4. Acionar "Salvar"
  5. Localizar o tópico de curso cadastrado no final da listagem

### 3.3.4 Caso de teste

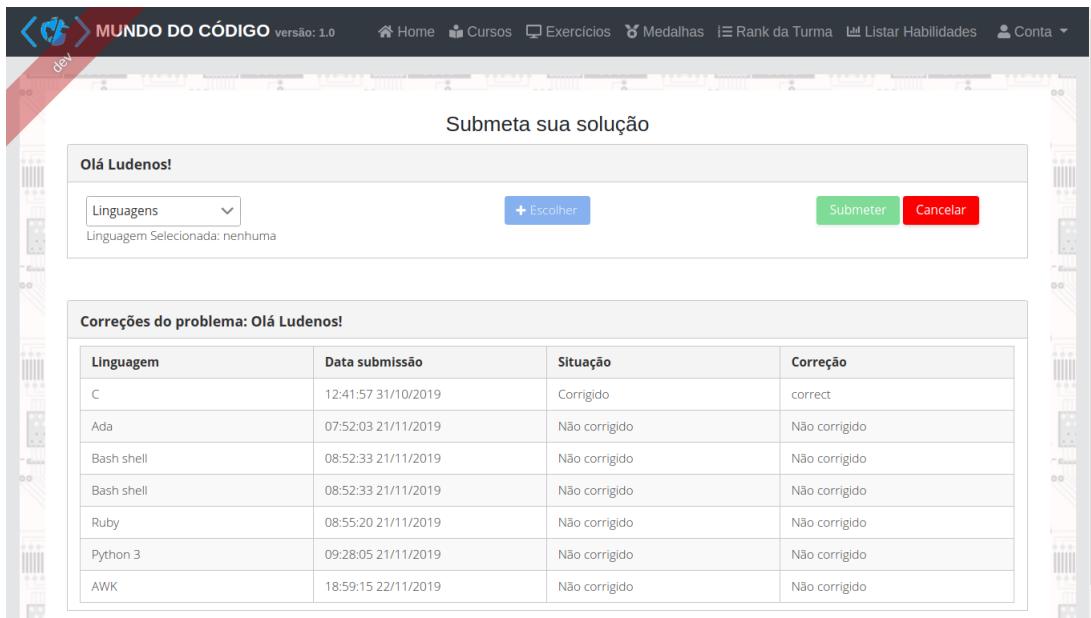
Tabela 4 – Caso de teste de sistema - Tópicos de curso

Caso de teste de sistema - Tópicos de curso									
1. Projeto	Mundo do Código								
2. Testador	André Marcelino de Souza Neves e Leonam Teixeira de Vasconcelos								
3. Identificação	TESTE01 - topico								
4. Data da Elaboração	23/11/2019								
5. Itens a testar	Validação dos campos e se o processo de criação é realizado corretamente.								
6. Procedimento de teste	Verificar se a criação de um tópico de curso é realizado corretamente.								
7. Entradas	Campo	Valores							
		T1	T2	T3	T4				
	Nome	Vazio	Grafos 2	Árvores 2	Árvores 2				
	Descrição	Texto	Grafos avançados	Árvores avançado	Árvores avançado				
	Ordem	1	8	8	arvore				
	Html	Vazio	<h1>Grafos Avançado</h1>	<h1>Árvores Avançado</h1>	<h1>Árvores Avançado</h1>				
	Pdf	Vazio	Vazio	Vazio	Vazio				
	Imagen	Vazio	Vazio	codigo.java	Vazio				
8. Saídas Esperadas	T1	Indicação dos campos obrigatórios destacados de vermelho							
	T2	Mensagem de tópico criado com sucesso							
	T3	Não conseguir inserir o arquivo							
	T4	Não conseguir inserir nenhum dados no campo							
9. Checagem Necessária	T1	Novo tópico não é gravado no banco							
	T2	Novo tópico criado no banco							
	T3	Arquivo é inserido no campo							
	T4	O campo não parece nenhum carácter a não ser números							
10. Resultado da aplicação do caso de teste	T1	Teste aprovado							
	T2	Teste aprovado							
	T3	Teste aprovado							
	T4	Teste aprovado							
Legenda	<b>Azul:</b> teste executado, erro encontrado e corrigido								
	<b>Verde:</b> Teste executado e nenhum erro encontrado.								
	<b>Vermelho:</b> teste executado, erro encontrado e aguardando correção.								

### 3.4 Caso de uso 2: Submissão de Exercício

O segundo caso de uso que apresentaremos a seguir é o "Submeter Solução". A descrição desse caso de uso é: "Recebe um arquivo de código fonte, manda as informações para o julgador e retorna as informações da submissão".

Figura 19 – Fotografias da tela de submissão.



**Fonte:** Autores.

Através da imagem 19 observamos a tela de Submissão. Nela é possível ver o "Drop-down" contendo várias linguagens para submissão. Além disso, observa-se o botão "Escolha", sua função é permitir ao usuário escolher um arquivo de seu computador e enviá-lo para correção, ele permanecerá desabilitado até que uma linguagem seja escolhida. Somado a isso, outros dois botões, "Submeter"(em verde) e "Cancelar"(em vermelho) estão presentes. O botão submeter permanecerá desabilitado até que um arquivo seja escolhido, enquanto o botão cancelar estará habilitado o tempo todo, para que o usuário possa cancelar suas escolhas a qualquer momento. Por fim, uma listagem com os arquivos submetidos, contendo data e hora da submissão, estado de correção (Corrigido ou não) e o resultado dos testes de resolução são apresentados na tabela na parte inferior da tela.

Essa tela, apesar de parecer simples, é a mais complexa de todas aquelas que foram produzidas nesse projeto. Isso ocorre, porque internamente ela possui diversos métodos, campos e validações obrigatórios para o bom funcionamento da funcionalidade que ela apresenta. O objetivo dos projetistas com essa tela é que sua função seja integrada a uma API de julgador online chamada de "DomJudge". Por conta das exigências desse julgador, as validações acima citadas são tão necessárias. O funcionamento básico desse julgador é esperar até que uma nova submissão seja feita. Quando isso ocorre, a API seleciona o arquivo que foi submetido e sobre ele executa vários casos de teste que buscam comprovar se o algoritmo está correto. Ao final dos testes, ela responde se o exercício foi aprovado ou não para que os resultados desse processo sejam mostrados na tabela. Para que esse processo

ocorra, deve-se informar à Api: A linguagem, o Arquivo de solução e vários outros campos que serão perpetuados através de duas tabelas especiais do banco de dados: "Exercícios Resolvidos" e "Exercícios Resolvidos Arquivo". Cada uma dessas tabelas possuem chaves estrangeiras obrigatórias, que exigem pelo menos um "Get"(busca) cada. Além disso, o arquivo submetido deve ser convertido em uma "String de base 64"(frequentemente utilizada para envio de arquivos na internet), antes que o método "create"(criação) seja chamado para salvar as tabelas preenchidas. Por conta dessas buscas, seleções e validações, justifica-se a escolha anterior de um dos métodos que validam os dados dessa tela e a necessidade da presença de um condicional com pelo menos 4 "ses" diferentes.

### 3.4.1 Fluxo de Submissão e Print Screen das telas

Para demonstrar como ocorre o fluxo de submissão, será apresentada abaixo uma sequencia de imagens com explicações que ilustrarão os procedimentos que devem ser feitos para submeter um arquivo ao julgamento.

Figura 20 – Tela inicial(Painel) do módulo aluno.



Fonte: Autores.

Na figura 20, vê-se o painel do módulo aluno. Essa é a primeira tela que o aluno verá ao entrar no módulo destinado a ele. Para fazer uma submissão ele deve selecionar o Comando Exercícios, marcado pelo retângulo azul.

Figura 21 – Tela de listagem de aluno

**Fonte:** Autores.

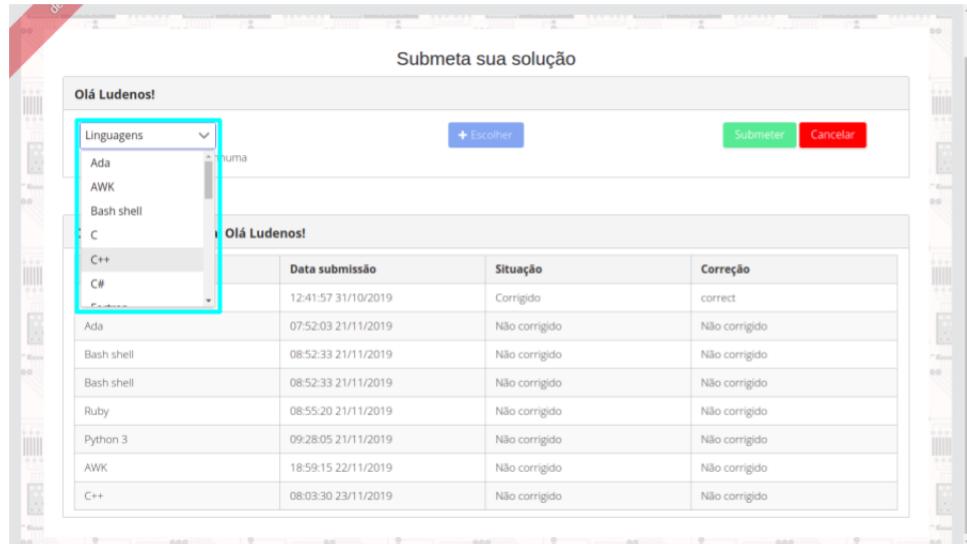
Ao clicar no botão "Exercícios", o aluno verá a tela de Listagem de exercícios, como pode-se observar através da figura 21. Nessa tela, o usuário poderá escolher qual exercício ele deseja resolver. Um exemplo de problema que pode ser escolhido é o marcado pelo retângulo em azul. Ao selecionar esse botão, um tela de visualização de exercício aparecerá. Nesta tela, o aluno selecionará o botão Submeter solução e a tela de submissão será visualizada.

Figura 22 – Seleção das linguagens

**Fonte:** Autores.

Estando na tela de submissão, o aluno poderá escolher qual linguagem deseja, através do "dropdown" que possui uma lista de linguagens. Essa lista pode ser observada através da imagem 23.

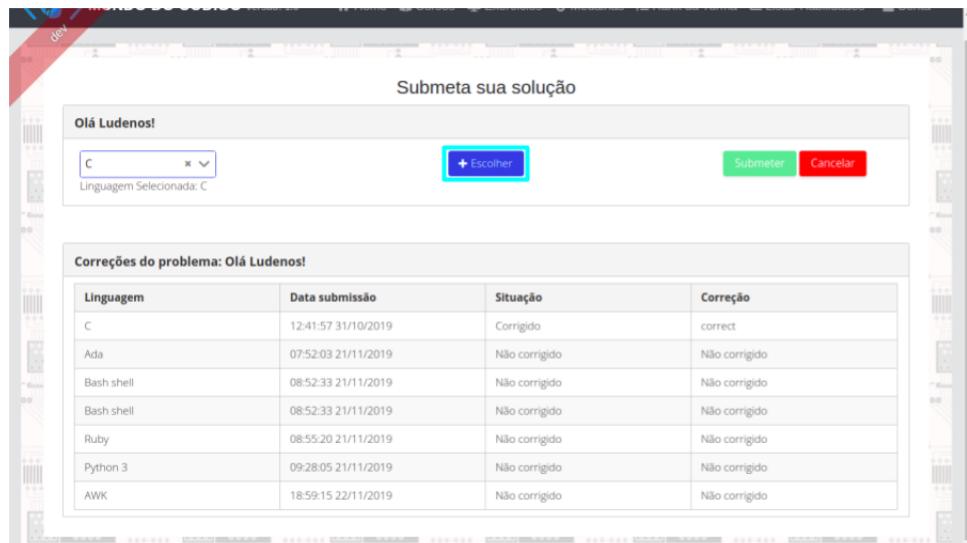
Figura 23 – Conteúdo do "Dropdown"



Fonte: Autores.

Quando a linguagem for escolhida, o comando "Escolher" azul ficará habilitado e poderá ser utilizado para o internauta selecionar seu arquivo para submissão e julgamento. Como observado na figura 24.

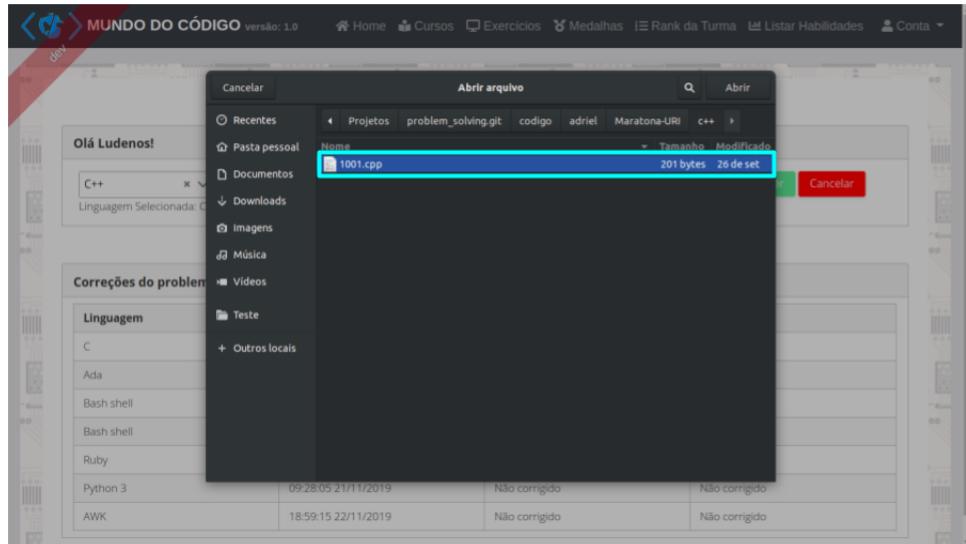
Figura 24 – Selecionando um arquivo



Fonte: Autores.

Ao clicar no comando "Escolher", o usuário poderá navegar pelas pastas e arquivos de seu computador e escolher um arquivo para submeter. Essa navegação é exemplificada pela figura 26.

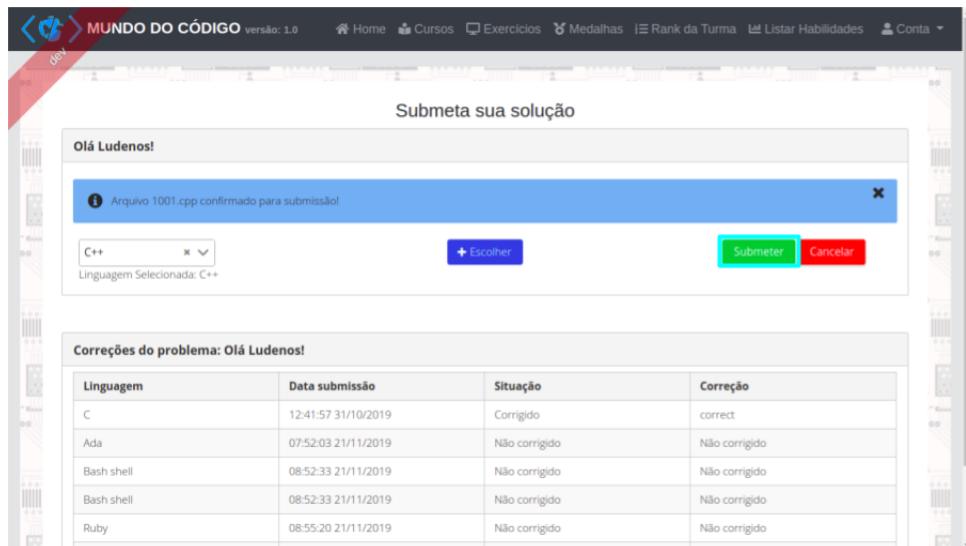
Figura 25 – Navegação interna



Fonte: Autores.

Após selecionado e submetido o arquivo, o comando cujo botão está verde, "Submeter", ficará habilitado e poderá ser utilizado para confirmar a submissão. Caso o usuário queira cancelar, basta clicar a qualquer momento no botão em vermelho "Cancelar" e será roteado ao problema original. A finalização desse processo pode ser observado na figura 26.

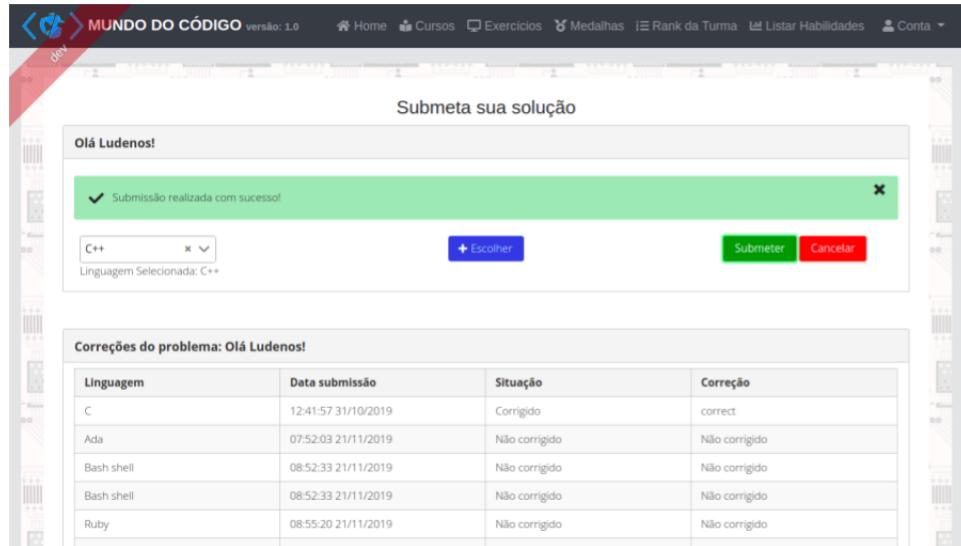
Figura 26 – Navegação interna



Fonte: Autores.

Quando isso ocorrer, uma mensagem de confirmação aparecerá como pode ser visto na imagem 27.

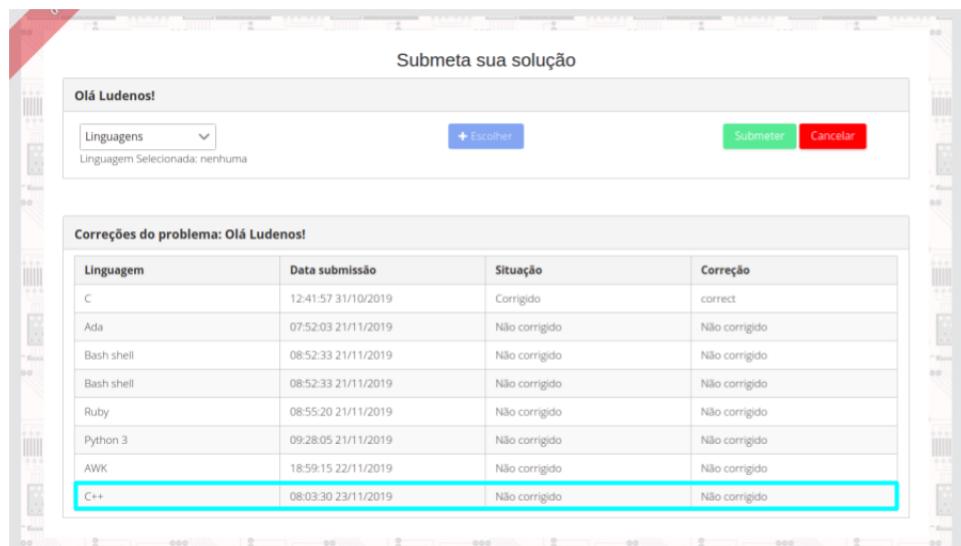
Figura 27 – Confirmação da submissão



**Fonte:** Autores.

Ao fim do processo, o aluno poderá confirmar sua submissão recarregando a página e checando na lista de submissões abaixo se sua submissão foi realmente perpetuada no banco de dados. Como feito na figura 28.

Figura 28 – Confirmação da submissão



**Fonte:** Autores.

### 3.4.2 Procedimento de teste

- Objetivo: Verificar se a submissão de exercício é feita corretamente.
- Fluxo:
  1. Acessar painel principal
  2. Acionar o botão "Exercícios"
  3. Acionar o botão "Submeter Solução"
  4. Escolher a Linguagem
  5. Selecionar um arquivo
  6. Acionar "Submeter"
  7. Localizar a submissão de curso cadastrado no final da tabela na parte inferior da página

Função do caso de teste: Possibilita ao aluno submeter um arquivo contendo um algoritmo de solução para o problema especificado juntamente com a escolha de uma linguagem. No caso de teste forneceu-se as seguintes informações:

- Linguagem de Desenvolvimento da Solução
- Arquivo com algoritmo de resolução do problema

Tabela 5 – Caso de teste de sistema - Submissão de exercício

Caso de teste de sistema - Submissão de exercício					
1. Projeto	Mundo do Código				
2. Testador	André Marcelino de Souza Neves e Leonam Teixeira de Vasconcelos				
3. Identificação	TESTE02 - submissao				
4. Data da elaboração	23/11/2019				
5. Itens a testar	Desabilitação dos comandos "Escolher" e "Submeter", Validação dos campos, além da inserção e da listagem correta.				
6. Procedimento de teste	Verificar se a submissão de um arquivo de solução é executada de forma correta.				
7. Entradas	Campo	Valores			
		T1	T2	T3	T4
	Linguagem	C++	Vazio	Ruby	Python 3
	Arquivo	Vazio	Nada	1001.rb	1001.py
8. Saídas Esperadas	T1	O comando "Submeter" permanecerá desabilitado.			
	T2	O comando "Escolher" permanecerá desabilitado.			
	T3	O arquivo deve ser submetido com sucesso, uma mensagem com o texto "Arquivo 1001.rb confirmado para submissão!" e ao recarregar a página os dados dessa submissão devem aparecer na tabela.			

	T4	O arquivo deve ser submetido com sucesso, uma mensagem com o texto "Arquivo 1001.py confirmado para submissão!" e ao recarregar a página os dados dessa submissão devem aparecer na tabela.
9. Checagem Necessária	T1	Verificar se os comando "Submeter" está realmente desabilitado.
	T2	Verificar se o comando "Escolher" está desabilitado.
	T3	Verificar se após o carregamento da página uma nova linha foi adicionada a tabela e se os dados são compatíveis com o problema recém submetido.
	T4	Verificar se após o carregamento da página uma nova linha foi adicionada a tabela e se os dados são compatíveis com o problema recém submetido.
10. Resultado da aplicação do caso de teste	T1	Teste aprovado
	T2	Teste aprovado
	T3	Teste aprovado
	T4	Teste aprovado
Legenda	<b>Azul:</b> teste executado, erro encontrado e corrigido	
	<b>Verde:</b> Teste executado e nenhum erro encontrado.	
	<b>Vermelho:</b> teste executado, erro encontrado e aguardando correção.	

## 4 Conclusão

Com esse trabalho, foi possível colocar em prática os conceitos e técnicas de testes. Ambas as partes do trabalho referem-se à conceitos importante de serem praticados no cotidiano de desenvolvimento de software.

A automação de testes unitário e de cobertura é essencial para garantir correta implementação de funções implementadas que realizam algum tipo de processamento específico. A existência dos testes automatizados ajudam a garantir que nenhuma modificação na implementação prejudique o correto funcionamento do sistema em casos específicos.

A realização de testes de sistemas é essencial para verificar se a usabilidade das funcionalidades é adequada para o usuário. Por meio dela, é possível verificar se o sistema realiza tratamento de ações inválidas realizadas pelo usuário, ou até mesmo conduz com fluidez o processo de uso do software, de forma a melhorar a experiência do usuário.

Portanto, é necessário avaliar tais práticas em projetos reais, para que o mínimo de qualidade seja garantida na entrega do produto final.

## Referências

Devmedia. *Testes de software - Teste Unitario - Code coverage*. 2011. Disponível em: <<https://www.devmedia.com.br/testes-de-software-teste-unitario-code-coverage/22286>>. Citado na página 6.

PRESSMAN, R. S. *Engenharia de software : uma abordagem profissional*. [s.n.]. ISBN 8580550440. Disponível em: <[https://books.google.com.br/books?id=y0rH9wuXe68C&dq=pressman+engenharia+de+software&hl=pt-BR&sa=X&ved=0ahUKEwj1j\\_61gYPmAhXEt1kKHYj7CLcQ6AEIOzAC](https://books.google.com.br/books?id=y0rH9wuXe68C&dq=pressman+engenharia+de+software&hl=pt-BR&sa=X&ved=0ahUKEwj1j_61gYPmAhXEt1kKHYj7CLcQ6AEIOzAC)>. Citado na página 7.

SOMMERVILLE, I. *Engenharia de software*. Pearson Prentice Hall, 2011. ISBN 8579361087. Disponível em: <<https://books.google.com.br/books?id=H4u5ygAACAAJ&dq=sommerville+engenharia+de+software&hl=pt-BR&sa=X&ved=0ahUKEwj1t-jE5YLmAhWwtVkKHaljBtsQ6AEIKjAA>>. Citado na página 6.