

**PROJETO 2022.01 - UNDERLYING  
Testes Estruturais (DTE)**

**Caixa-Branca**

**Versão 0.2**

**Equipe de Projeto Underlyng:**

**Bruno Brandão Borges - 2018014331**

**Ivan Leoni Vilas Boas - 2018009073**

**Leonardo Rodrigo de Sousa - 2018015965**

**Lucas Tiense Blazzi - 2018003310**

**Thiago Marcelo Passos - 2018002850**

**Wesley Alexandre de Almeida Gomes - 2018005806**



**IMC - Instituto de Matemática e Computação**

Av. BPS, 1303 - Caixa postal 50 - 37500-903

Itajubá - MG - Brasil Telefone: 35-3629-1135

E-mail:[imc@unifei.edu.br](mailto:imc@unifei.edu.br)

Revisões do Documento

Revisões são melhoramentos na estrutura do documento e também no seu conteúdo. O objetivo primário desta tabela é a fácil identificação da versão do documento. Toda modificação no documento deve constar nesta tabela.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data** | **Versão** | | **Descrição** | **Autor** |
| 26/06/2022 | 0.1 | Elaboração do documento de teste | | Ivan |
| 16/07/2022 | 0.2 | Cobertura dos testes | | Lucas |

Auditorias do Documento

Auditorias são inspeções conduzidas o SEPG – Software Engineer Process Group (Grupo de Engenharia de Processo de Software), e tem por objetivo garantir uma qualidade mínima dos artefatos gerados durante o processo de desenvolvimento. Essa tabela pode ser utilizada também pelo GN – Gerente da Área de Negócio com o objetivo de documentar a viabilidade do mesmo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data** | | **Versão** | | **Descrição** | **Autor** |
| 27/06/2022 | 0.1 | | Revisão do documento de teste | | Lucas |
| 17/07/2022 | | 0.2 | | Revisão do documento de teste | Ivan |

**Índice de Ilustrações**

[Figura 1 - Cobertura de testes – Serviço de Estratégias 11](#_Toc109163318)

[Figura 2 - Cobertura de testes – Serviço de Opções 12](#_Toc109163319)

[Figura 3 - Teste para Visualização das Estratégias 17](#_Toc109163320)

[Figura 4 - GFC para edição da estratégia 20](#_Toc109163321)

[Figura 5 - Função de validação do email 40](#_Toc109163322)

[Figura 6 - GFC para validação do Email 40](#_Toc109163323)

[Figura 7 - Análise de todos os nós para validar email 41](#_Toc109163324)

[Figura 8 - Função de validação do nome do usuário 42](#_Toc109163325)

[Figura 9 - FGC para validação do nome do usuário 43](#_Toc109163326)

[Figura 10 - Análise de todos os nós para nome do usuário 43](#_Toc109163327)

[Figura 11 - Função de cálculo de Payoff 45](#_Toc109163328)

[Figura 12 - Função lambda\_handler 45](#_Toc109163329)

[Figura 13 - GFC para Cálculo de Payoff 46](#_Toc109163330)

[Figura 14 - Análise de todos os nós para validar o cálculo de Payoff 46](#_Toc109163331)

[Figura 15 - Código para a validação do campo de Email 50](#_Toc109163332)

[Figura 16- FGC para validar campo de email 51](#_Toc109163333)

[Figura 17 - Análise de todos os nós para validar campo Email 52](#_Toc109163334)

[Figura 18 - Código para a validação do campo de Senha 54](#_Toc109163335)

[Figura 19 - GFC do campo de senha 55](#_Toc109163336)

[Figura 20 - Análise de todos os nós para o campo Senha 56](#_Toc109163337)

[Figura 21 - GFC para busca de opções 58](#_Toc109163338)

[Figura 22 - Análise de todos os nós para busca de opções 58](#_Toc109163339)

[Figura 23 - Busca de opção (backend) 59](#_Toc109163340)

[Figura 24 - Análise de todos os nós para busca de opção backend 60](#_Toc109163341)

[Figura 25 - Função de Search 62](#_Toc109163342)

[Figura 26 - Análise de todos os nós para Search 62](#_Toc109163343)

[Figura 27 - Análise de todas as arestas para validar o cálculo de payoff do Backend 62](#_Toc109163344)

[Figura 28 - Função Health 63](#_Toc109163345)

[Figura 29 - GFC para cálculo de health check 63](#_Toc109163346)

[Figura 30 - Análise de todos os nós para validar o health check 64](#_Toc109163347)

[Figura 31 - Função de validação da inserção 65](#_Toc109163348)

[Figura 32 – CFG de validação de inserção 65](#_Toc109163349)

[Figura 33 – Análise de todos os nós para validar inserção 66](#_Toc109163350)

**Índice de Tabelas**

[Tabela 1 - Responsabilidade dos testes estruturais 9](#_Toc109163351)

[Tabela 2 - GFC para visualização da estratégia 13](#_Toc109163352)

[Tabela 3 - Análise do GFC para teste em todos os nós da leitura da estratégia 14](#_Toc109163353)

[Tabela 4 - Teste em todos os nós da leitura da estratégia 15](#_Toc109163354)

[Tabela 5 - Análise do GFC teste de todas as arestas na leitura da estratégia 15](#_Toc109163355)

[Tabela 6 - Teste em todos as arestas da leitura da estratégia 16](#_Toc109163356)

[Tabela 7 - Teste em todos os caminhos da visualização da estratégia 16](#_Toc109163357)

[Tabela 8 - Teste em todos os nós para atualização de estratégia 24](#_Toc109163358)

[Tabela 9 - Teste em todas as arestas da atualização da estratégia 31](#_Toc109163359)

[Tabela 10 - Teste em todos os caminhos para atualização de estratégia 39](#_Toc109163360)

[Tabela 11 – Teste para todas as arestas para validar email 41](#_Toc109163361)

[Tabela 12 - Análise de todas as arestas para nome do usuário 44](#_Toc109163362)

[Tabela 13 - Teste de todas as arestas para validar o cálculo de Payoff 49](#_Toc109163363)

[Tabela 14 - Análise de todas as arestas para validar campo de email 53](#_Toc109163364)

[Tabela 15 - Análise de todas as arestas do GFC para o campo senha 57](#_Toc109163365)

[Tabela 16 - Análise de todas as arestas no GFC para busca de opções frontend 59](#_Toc109163366)

[Tabela 17 - Análise de todas as arestas para validar o cálculo de payoff backend 61](#_Toc109163367)

[Tabela 18 - Análise de todas as arestas para validar o health 64](#_Toc109163368)

[Tabela 19 - Teste para todas as arestas para validar inserção 66](#_Toc109163369)

**Sumário**

[1. Introdução 7](#_Toc109163436)

[1.1 Técnica do teste estrutural 7](#_Toc109163437)

[1.2 Responsabilidades de testes da equipe de projeto 8](#_Toc109163438)

[2. COBERTURA DOS TESTES 9](#_Toc109163439)

[3. Teste da visualização das estratégias 13](#_Toc109163440)

[3.1 Criação do GFC para visualização da estratégia 13](#_Toc109163441)

[3.2 Teste Caixa Branca: Técnica do caminho para a leitura das estratégias 13](#_Toc109163442)

[3.3 Teste em todos os nós para a leitura das estratégias 14](#_Toc109163443)

[2.3. Teste em todas as arestas para visualizar estratégia 15](#_Toc109163444)

[2.4. Teste em todos os caminhos para visualizar estratégia 16](#_Toc109163445)

[3. Teste da edição das estratégias 17](#_Toc109163446)

[3.1. Criação do GFC para edição da estratégia 18](#_Toc109163447)

[3.2. Teste em todos os nós para atualização de estratégia 20](#_Toc109163448)

[3.3. Teste em todos as arestas para atualização de estratégia 24](#_Toc109163449)

[3.4. Teste em todos os caminhos para atualização de estratégia 31](#_Toc109163450)

[4. Teste de cadastro de usuário 39](#_Toc109163451)

[4.1. Teste do Campo de Email 39](#_Toc109163452)

[4.1.1. Função de validação do email 39](#_Toc109163453)

[4.1.2. GFC para validação do Email 40](#_Toc109163454)

[4.1.3. Análise de todos os nós do GFC para validação do Email 41](#_Toc109163455)

[4.1.4. Análise de todas as arestas para validar email 41](#_Toc109163456)

[4.2. Teste do Campo de nome de usuário 42](#_Toc109163457)

[4.2.1. Função de validação do nome do usuário 42](#_Toc109163458)

[4.2.2. FGC para validação do nome do usuário 42](#_Toc109163459)

[4.2.3. Análise de todos os nós para nome do usuário 43](#_Toc109163460)

[4.2.4. Análise de todas as arestas para nome do usuário 43](#_Toc109163461)

[5. Teste de calculo de payoff 44](#_Toc109163462)

[5.1. Função de cálculo de Payoff 44](#_Toc109163463)

[5.2. GFC para Calculo de Payoff 45](#_Toc109163464)

[5.3. Análise de todos os nós para validar o cálculo de Payoff 46](#_Toc109163465)

[5.4. Análise de todas as arestas para validar o cálculo de Payoff 47](#_Toc109163466)

[6. Teste de Autenticação de usuário 49](#_Toc109163467)

[6.1. Validação do campo de email 50](#_Toc109163468)

[6.1.1. Código para a validação do campo de Email 50](#_Toc109163469)

[6.1.2. FGC para validar campo de email 51](#_Toc109163470)

[6.1.3. Análise de todos os nós para validar campo Email 52](#_Toc109163471)

[6.1.4. Análise de todas as arestas para validar campo de email 53](#_Toc109163472)

[6.2. Validação do campo de Senha 53](#_Toc109163473)

[6.2.1. Código para a validação do campo de Senha 53](#_Toc109163474)

[6.2.2. GFC do campo de senha 55](#_Toc109163475)

[6.2.3. Análise de todos os nós para o campo Senha 56](#_Toc109163476)

[6.2.4. Análise de todas as arestas do GFC para o campo senha 57](#_Toc109163477)

[7. Teste da busca de opções Frontend 57](#_Toc109163478)

[7.1. Funções de controle da validação de busca de opções frontend 57](#_Toc109163479)

[7.2. GFC para busca de opções frontend 58](#_Toc109163480)

[7.3. Análise de todos os nós para busca de opções frontend 58](#_Toc109163481)

[7.4. Análise de todas as arestas no GFC para busca de opções frontend 59](#_Toc109163482)

[8. Teste da busca de opções backend 59](#_Toc109163483)

[8.1. Função de busca de opção backend 59](#_Toc109163484)

[8.2. Análise de todos os nós para busca de opção backend 60](#_Toc109163485)

[8.3. Análise de todas as arestas para validar o cálculo de payoff backend 60](#_Toc109163486)

[8.4. Teste Search 61](#_Toc109163487)

[8.5. Função de Search 61](#_Toc109163488)

[8.6. Análise de todos os nós para Search 62](#_Toc109163489)

[8.7. Análise de todas as arestas para validar o cálculo de payoff do Backend 62](#_Toc109163490)

[9. Teste de health ckeck 63](#_Toc109163491)

[9.1. Função Health 63](#_Toc109163492)

[9.2. GFC para cálculo de health check 63](#_Toc109163493)

[9.3. Análise de todos os nós para validar o health check 64](#_Toc109163494)

[9.4. Análise de todas as arestas para validar o health 64](#_Toc109163495)

[10. Teste de inserção de opção fictícia 64](#_Toc109163496)

[10.1. Teste de validade de inserção 64](#_Toc109163497)

[10.1.1. Função de validação de inserção 65](#_Toc109163498)

[10.1.2. GFC para validação de inserção 65](#_Toc109163499)

[10.1.3. Análise de todos os nós do GFC para validação de inserção 66](#_Toc109163500)

[10.1.4. Análise de todas as arestas para validar inserção 66](#_Toc109163501)

# Introdução

Esse documento apresenta os testes estruturais que são fundamentais para garantir a qualidade do sistema. O teste estrutural, ou de caixa-branca, é projetado em função da estrutura interna do sistema, e por isso permite uma verificação mais precisa do funcionamento do *software.* Realiza a validação do código-fonte da aplicação, bem como dos diferentes algoritmos e estruturas de dados. Em suma, o teste seleciona diferentes valores de entrada, para examinar cada um dos possíveis fluxos de execução do programa e verificar se os valores de saída estão retornando corretamente. Este tipo de teste é desenvolvido analisando o código-fonte e elaborando casos de teste que cubram as funcionalidades do componente de *software*. Seu objetivo é verificar os fluxos de execução dentro de funções, classes, módulos, entre outros, bem como pode ser aplicado para validar os fluxos entre as unidades durante a integração e/ou entre subsistemas.

Essa técnica é vista como complementar à técnica funcional e informações obtidas pela aplicação desses critérios têm sido consideradas relevantes para as atividades de manutenção, depuração e para a confiabilidade de *software*. A técnica de teste de caixa-branca é recomendada para as fases de teste de unidade e teste de integração, cuja responsabilidade principal fica a cargo dos desenvolvedores do sistema, que por sua vez conhecem bem o código produzido.

Por meio de testes estruturais é possível: Assegurar que todos os caminhos independentes de cada módulo, programa ou método sejam executados pelo menos uma vez; verificar todas as decisões lógicas de verdadeiro e falso; executar todos os loops em seus limites operacionais; executar as estruturas de dados internas para garantir a validade. Caso ocorra alguma alteração antes da etapa de implementação, os testes normalmente terão que ser refeitos.

O Teste funcional procura erros nos resultados do programa, enquanto o Teste estrutural procura erros na construção do programa (chamada de métodos, troca de dados, interações entre unidades) que podem afetar a saída do programa. Assim sendo, o teste de caixa-branca pode ser considerado um dos mais importantes tipos de testes que se aplicam ao software, tendo como resultado a [diminuição no número de erros no sistema](https://www.treinaweb.com.br/blog/importancia-dos-testes-de-software-na-qualidade-do-sistema/) e, portanto, contribuindo com uma maior qualidade e confiabilidade.

## Técnica do teste estrutural

A técnicas a ser aplicadas aos testes estruturais do sistema será o teste do Caminho Básico que é uma técnica proposta por Thomas J. McCabe, em 1976, que permite ter uma noção da complexidade lógica de um projeto e, posteriormente, usar essa medida como um guia para a definição de um conjunto básico de caminhos de execução. A ideia é realizar casos de testes a partir de um determinado conjunto de caminhos independentes. Para obter este conjunto, é construído um fluxograma e sua complexidade ciclomática é calculada.

As etapas para aplicar esta técnica são: Desenhar o fluxograma a partir do código-fonte; calcular a complexidade ciclomática do gráfico e determinar um conjunto básico de caminhos independentes. Por fim são preparados os casos de teste, que requerem a execução de cada um dos caminhos. Para aplicar a técnica de caminho básico, deve ser usado uma notação simples para a representação do fluxo de controle. O fluxograma costuma ser composto por 3 componentes fundamentais, que ajudam a preparar, entender e fornecer informações, para assegurar que o trabalho está sendo executado corretamente. Os componentes são os Nós que representam uma sequência de processos ou uma declaração de decisão; as Setas que representam o fluxo de controle. Uma seta deve sempre terminar em um nó, mesmo que este não passe nenhuma instrução do procedimento; e as Regiões que são as áreas delimitadas por setas e nós. A área externa do gráfico também é incluída, contando como mais uma região.

  A complexidade ciclomática é uma métrica de software extremamente útil, pois fornece uma medida quantitativa da complexidade lógica de um programa. Está métrica mede a quantidade de diferentes fluxos de execução que o código pode ter, ou seja, quantos *ifs-then-else, while, for, switch*, entre outros, há no código-fonte. Quanto maior a complexidade ciclomática, mais complicado o código será de ler, entender, modificar, manter e, consequentemente, será mais caro.

Um caminho independente é qualquer rota no programa que introduz pelo menos um novo conjunto de instruções de processo, ou uma condição, em relação aos caminhos existentes. Em termos de diagrama de fluxo, consiste em pelo menos uma seta que não foi percorrida antes da definição do caminho. Ao identificar os diferentes caminhos de um programa a ser testado, deve-se levar em consideração que cada nova rota deve ter novas condições em relação às já existentes.

Este projeto utilizou as seguintes técnicas para o teste estrutural:

* Técnica de todos os caminhos
* Técnica de todos os nós
* Técnica de todas as arestas

## Responsabilidades de testes da equipe de projeto

A seguir segue as responsabilidades definidas para os testes estruturais do sistema Underlying:

|  |  |  |
| --- | --- | --- |
| **Testes Funcionais do****Sistema Underlying** | | |
| **Teste modulo** | **Descrição** | **responsável** |
| Editar estratégia de opções  (backend) | Realiza teste para a edição dos dados de entrada referentes a uma estratégia de opções no banco de dados DynamoDB | Lucas |
| Leitura das estratégias de opções  (backend) | Realiza o teste para leitura/visualização da estratégia de opções (backend) no banco DynamoDB | Ivan |
| Cálculo de Payoff (backend) | Realiza os testes para Cálculo de Payoff em Estratégias | Bruno Brandão Borges |
| HEALTH CHECK | Utilizado para pingar na cloud functions |
| Busca de Opções  (Backend) | Testa o retorno da busca de opções. |
| Busca de Opções  (Frontend) | Testa o retorno da busca de opções. | Wesley |
| Cadastro de usuário (Frontend) | Realiza os testes dos dados cadastrais dos usuários antes de salvas os dados | Leonardo |
| Inserção de opção fictícia  (Frontend) | Realiza o teste para o cadastro de opção fictícia |
| Autenticação do usuário(frontend) | Realiza os testes das informações inseridas ao realizar o login | Thiago |

Tabela 1 - Responsabilidade dos testes estruturais

# COBERTURA DOS TESTES

Após a implementação dos cenários de testes que serão abordados no decorrer desse documento, foi possível dividir a execução dos testes de acordo com a função no sistema (frontend e backend) já que eles foram desenvolvidos por ferramentas diferentes, PyTest e Jest respectivamente. No caso do backend, os testes foram elaborados individualmente nos micro serviços, podendo ser avaliado, então, a cobertura individual de cada um deles, já que estão isolados na etapa de deployment. A partir da implementação dos testes foi possível atingir os resultados:

* Backend – Serviço de opções – 79% de cobertura de código
* Backend – Serviço de estratégias – 97% de cobertura de código
* Frontend – Totalidade – todas as validações de formulários

Esses resultados podem ser visualizados nas figuras a seguir em questão de cobertura:

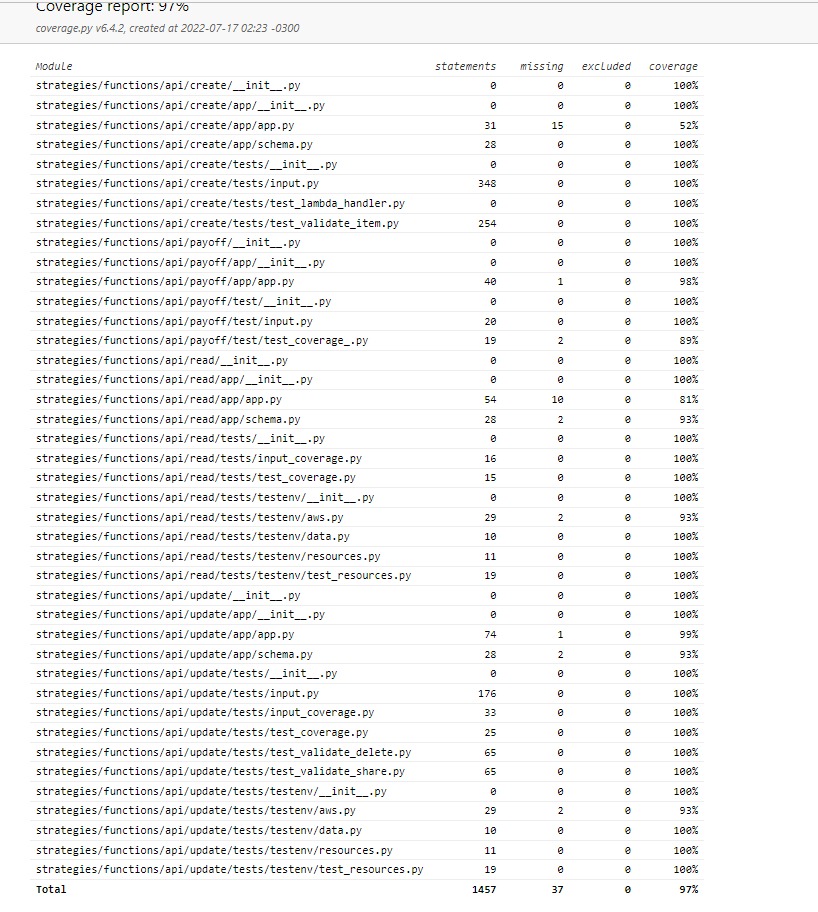


Figura 1 - Cobertura de testes – Serviço de Estratégias

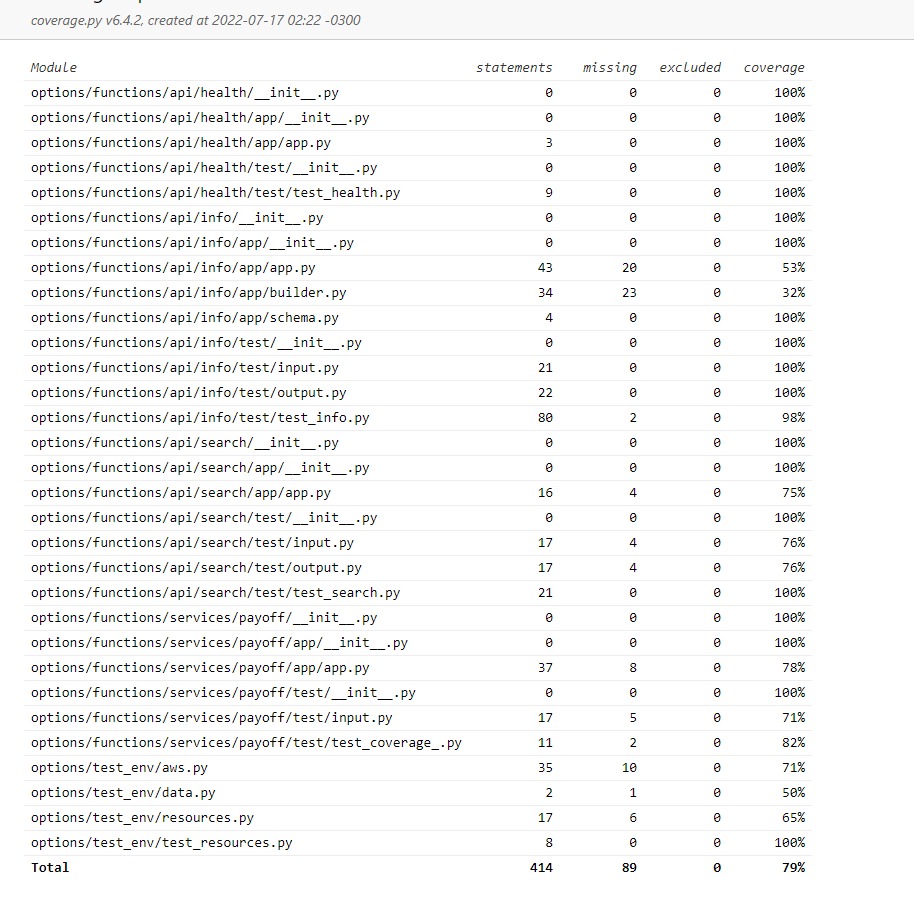


Figura 2 - Cobertura de testes – Serviço de Opções

# Teste da visualização das estratégias

A classe de atualização da interface (class UpdateInterface) responsável pela visualização da estratégia, ou seja, realiza a leitura das estratégias de opções do usuário do sistema Underlying. Os métodos a seguir formar a classe de atualização da interface:

* **get\_strategy(self, evento):** Realiza a busca da estratégia do usuário
* **get\_shared\_strategies(self, kwargs):** Realiza a busca da estratégia compartilhada pelo usuário
* **general\_get(self**): direcionar pra função certa get\_strategy ou get\_shared\_strategies
* **lambda\_handler (evento, contexto):** retorna corretamente os dados das estratégias do usuário ou mensagem de erro acerca da busca das estratégias.

## Criação do GFC para visualização da estratégia

Na Tabela a seguir é apresentado o GFC para a realização da leitura das estratégias criadas e compartilhadas pelos usuários.

|  |  |
| --- | --- |
| **Grafo de Fluxo de Controle (GFC)** | |
| Onde:  G1: Classe Update Interface  G2: Método general\_get  G3: Método get\_strategy  G6: Método get\_shared\_strategies  G4: Método lambda\_handler que retorna a estratégia caso encontrada  G5: Método lambda\_handler que retorna mensagem de erro caso não seja possível encontrar a estratégia |  |

Tabela 2 - GFC para visualização da estratégia

## Teste Caixa Branca: Técnica do caminho para a leitura das estratégias

A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais

1. Todos os nós
2. Todas as arestas
3. Todos os caminhos

## Teste em todos os nós para a leitura das estratégias

A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todos os nós:

|  |  |
| --- | --- |
| **Análise do GFC para Teste em todos os nós** | |
| Onde :  G1: Classe Update Interface  G2: Método general\_get  G3: Método get\_strategy  G6: Método get\_shared\_strategies  G4: Método lambda\_handler que retorna a estratégia caso encontrada  G5: Método lambda\_handler que retorna mensagem de erro caso não seja possível encontrar a estratégia  Caminhos:  CN1: G1-G2-G3-G4  CN2: G1-G2-G3-G5  CN3: G1-G2-G6-G4 |  |

Tabela 3 - Análise do GFC para teste em todos os nós da leitura da estratégia

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **CAMINHO** | **CASO** | **SAIDA** |
| CN1 | G1-G2-G3-G4 | id = 18199ad0-47fb-4592-8fb2-008726efab95 | assert 200 == 200  id valido  (Estratégia Encontrada) |
| CN2 | G1-G2-G3-G5 | id=1234 | id valido  (Estratégia não Encontrada) |
| CN3 | G1-G2-G6-G4 | vazio | assert 200 == 200  (Estratégia Encontrada) |

Tabela 4 - Teste em todos os nós da leitura da estratégia

## Teste em todas as arestas para visualizar estratégia

A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todas as arestas:

|  |  |
| --- | --- |
| **Análise do GFC para Teste em todas as arestas** | |
| Onde :  G1: Classe Update Interface  G2: Método general\_get  G3: Método get\_strategy  G6: Método get\_shared\_strategies  G4: Método lambda\_handler que retorna a estratégia caso encontrada  G5: Método lambda\_handler que retorna mensagem de erro caso não seja possível encontrar a estratégia  Caminhos:  CA1:A1-A2-A3  CA2:A1-A2-A4  CA3:A1-A5-A6 |  |

Tabela 5 - Análise do GFC teste de todas as arestas na leitura da estratégia

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **CAMINHO** | **CASO** | **SAIDA** |
| CA1 | A1-A2-A3 | id = 18199ad0-47fb-4592-8fb2-008726efab95 | assert 200 == 200  id valido  (Estratégia Encontrada) |
| CA2 | A1-A2-A4 | id= %4esFD12 | assert 500 == 500  id invalido  (Estratégia não Encontrada) |
| CA3 | A1-A5-A6 | vazio | assert 200 == 200  (Estratégia Encontrada) |

Tabela 6 - Teste em todos as arestas da leitura da estratégia

## Teste em todos os caminhos para visualizar estratégia

A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todas os caminhos:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **CAMINHO** | **CASO** | **SAIDA** |
| CC1 | **G1-G2-G3-G4** | id = 18199ad0-47fb-4592-8fb2-008726efab96 | assert 200 == 200  id valido  (Estratégia Encontrada) |
| CC2 | **G1-G2-G3-G4-G5** | NE (Não executável) |  |
| CC3 | **G1-G2-G3-G5** | id= hrtrd122-456 | assert 500 == 500  id invalido  (Estratégia não Encontrada) |
| CC4 | **G1-G2-G3-G6-G4** | NE (Não executável) |  |
| CC5 | **G1-G2-G6-G4** | vazio | assert 200 == 200  (Estratégia Encontrada) |

Tabela 7 - Teste em todos os caminhos da visualização da estratégia

A figura a seguir apresenta a execução do teste para leitura/visualização da Estratégia:

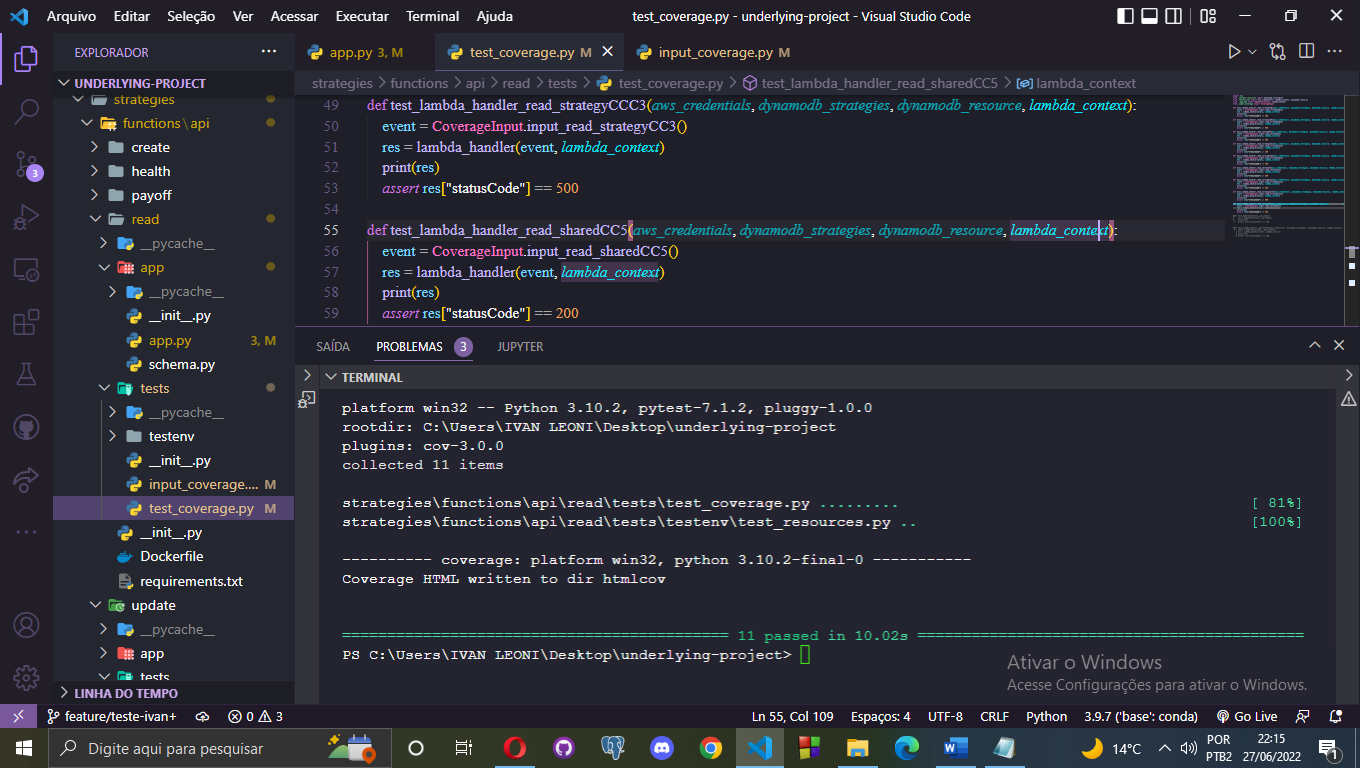


Figura 3 - Teste para Visualização das Estratégias

# Teste da edição das estratégias

A classe de atualização da interface (class UpdateInterface) responsável pela atualização da estratégia, ou seja, realiza a atualização das estratégias de opções do usuário do sistema Underlying, com base em sua entrada. Nesse cenário, temos a possibilidade de compartilhar uma estratégia, remover uma estratégia ou editar os atributos de uma estratégia. Os métodos a seguir formar a classe de atualização da interface:

* **\_\_get\_body():** carrega o corpo da requisição da API no formato adequado a ser manipulado.
* **\_\_get\_method():** retorna o método que será chamado a partir de sua identificação no path da chamada da API.
* **prepare\_item(item):** serializa o objeto recebido para conformidade com as interfaces de atualização do DynamoDB.
* **deserialize\_item(item):** deserializa o objeto recebido como resposta da atualização da estratégia no DynamoDB.
* **validate\_strategy (strategy):** valida individualmente a entrada de cada opção que compõe a estratégia, caso a necessidade de sua atualização.
* **validate\_share (share):** valida o payload recebido para a realização do compartilhamento da estratégia.
* **validate\_delete(delete):** valida o payload recebido para a realização da remoção da estratégia.
* **update\_item(self, item):** cria a conexão com a base de dados (DynamoDB) e define a hora de atualização do objeto, em seguida fazendo a atualização dele na base a partir da serialização prévia requerida, e retornando o novo objeto gerado a partir da deserialização dos dados.
* **share(self, event):** monta o payload referente ao compartilhamento da estratégia, fazendo a validação da entrada pela função de validação de compartilhamento e em seguida executando a função de atualização do objeto na base de dados.
* **delete(self, event):** monta o payload referente a remoção da estratégia, fazendo a validação da entrada pela função de validação de remoção e em seguida executando a função de atualização do objeto na base de dados.
* **update(self, event):** monta o payload para a atualização da estratégia, podendo ser uma atualização de nome ou de composição, no caso da alteração ser de composição, o objeto é validado pelo validador de estratégia, e em seguida é chamada a função de atualização do objeto na base de dados.
* **general\_update (self**): direcionar para a função correta que realizará a operação, de acordo com o path invocado na API, podendo ser update, share ou delete.
* **lambda\_handler (event, context):** instancia o evento recebido pela API para a classe de atualização, iniciando o fluxo de atualização, sendo responsável também pelo retorno da resposta a API.

## Criação do GFC para edição da estratégia

Na Tabela a seguir é apresentado o GFC para a realização da edição das estratégias criadas pelos usuários.

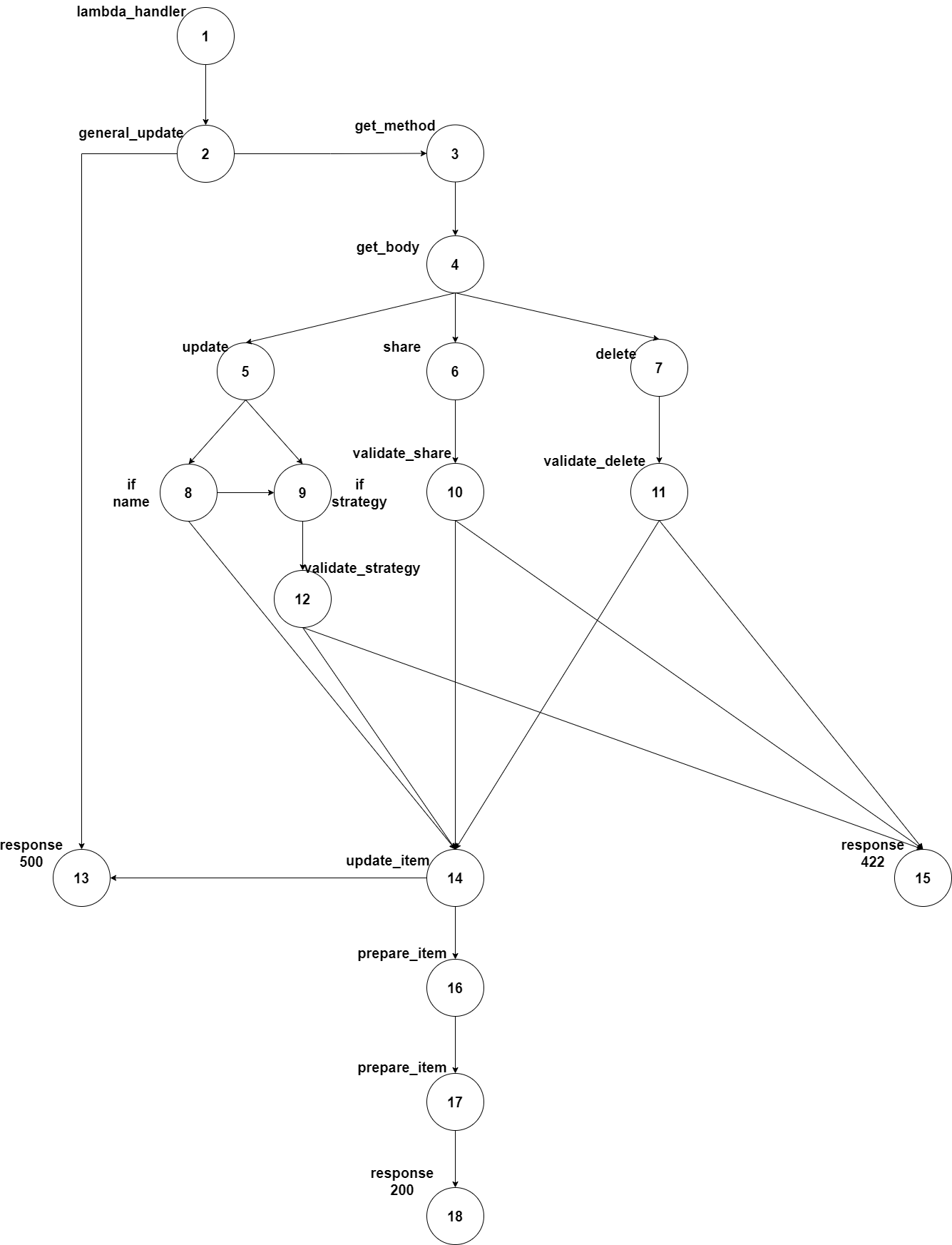


Figura 4 - GFC para edição da estratégia

## Teste em todos os nós para atualização de estratégia

A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todos os nós:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **CAMINHO** | **CASO** | **SAIDA** |
| CC1 | **G1-G2-G3-G4-G5-G8-G9-G12-**  **-G14-G16-G17-G18** | rawPath = “/v1/update”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG",  "close\_price": 1.23,  "contracts": 1,  "type": "PUT"  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "SHORT",  "close\_price": 1.23,  "contracts": 1,  "type": "CAll"  }  ]  }” | assert 200 == 200  (Objeto atualizado) |
| CC2 | **G1-G2-G3-G4-G6-G10-**  **-G14-G16-G17-G18** | rawPath = “/v1/share”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  “shared”: True  }” | assert 200 == 200  (Objeto compartilhado) |
| CC3 | **G1-G2-G3-G4-G7-G11-**  **-G14-G16-G17-G18** | rawPath = “/v1/delete”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  “deleted”: True  }” | assert 200 == 200  (Objeto removido) |
| CC4 | **G1-G2-G3-G2-G13** | rawPath = “/v1/invalid” | assert 500 == 500  (Erro – Rota não encontrada) |
| CC5 | **G1-G2-G3-G4-G6-G10-G15** | rawPath = “/v1/share”  body = “{  "id": "1234"  }” | assert 500 = 500  (Entrada inválida) |

Tabela 8 - Teste em todos os nós para atualização de estratégia

## Teste em todos as arestas para atualização de estratégia

A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todos as arestas:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **CAMINHO** | **CASO** | **SAIDA** |
| CC1 | **1-2-3-4-5-6-9-12-14-16-17-18** | rawPath = “/v1/update”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG",  "close\_price": 1.23,  "contracts": 1,  "type": "PUT"  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "SHORT",  "close\_price": 1.23,  "contracts": 1,  "type": "CAll"  }  ]  }” | assert 200 == 200  (Objeto atualizado) |
| CC2 | **1-2-3-4-6-10-14-16-17-18** | rawPath = “/v1/share”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  “shared”: True  }” | assert 200 == 200  (Objeto compartilhado) |
| CC3 | **1-2-3-4-7-11-14-16-17-18** | rawPath = “/v1/delete”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  “deleted”: True  }” | assert 200 == 200  (Objeto removido) |
| CC4 | **1-2-13** | rawPath = “/v1/invalid” | assert 500 == 500  (Erro – Rota não encontrada) |
| CC5 | **1-2-3-4-6-10-15** | rawPath = “/v1/share”  body = “{  "id": "1234"  }” | assert 500 = 500  (Entrada inválida) |

|  |  |  |  |
| --- | --- | --- | --- |
| CC6 | **1-2-3-4-5-9-12-14-15-16-17-18** | rawPath = “/v1/share”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG"  "close\_price": 1.23,  "contracts": 1,  "type": "PUT"  }  ]}” | assert 200 = 200  Estratégia atualizada |
| CC7 | **1-2-3-4-5-8-14-15-16-17-18** | rawPath = “/v1/update”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name”} | assert 200 = 200  Estratégia atualizada |
| CC8 | **1-2-3-4-5-8-12-15** | rawPath = “/v1/share”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG"  "close\_price": 1.23,  "contracts": 1,  "type": "INVALID"  }  ]}” | Assertion Error  Erro na edição |
| CC9 | **1-2-3-4-7-11-15** | rawPath = “/v1/delete”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  “deleted”: “INVALID”  }” | Assertion Error  Erro na edição |

Tabela 9 - Teste em todas as arestas da atualização da estratégia

## Teste em todos os caminhos para atualização de estratégia

A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todos os caminhos:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **CAMINHO** | **CASO** | **SAIDA** |
| CC1 | **1-2-3-4-5-8-14-15-16-17-18** | rawPath = “/v1/update”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name”} | assert 200 = 200  Estratégia atualizada |
| CC2 | **1-2-3-4-5-8-12-15** | rawPath = “/v1/share”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG"  "close\_price": 1.23,  "contracts": 1,  "type": "INVALID"  }  ]}” | Assertion Error  Erro na edição |
| CC3 | **1-2-3-4-7-11-15** | rawPath = “/v1/delete”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  “deleted”: “INVALID”  }” | Assertion Error  Erro na edição |
| CC4 | **1-2-3-4-5-9-12-14-15-16-17-18** | rawPath = “/v1/share”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG"  "close\_price": 1.23,  "contracts": 1,  "type": "PUT"  }  ]}” | assert 200 = 200  Estratégia atualizada |
| CC5 | **1-2-3-4-5-6-9-12-14-16-17-18** | rawPath = “/v1/update”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  "name": "Random Name",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG",  "close\_price": 1.23,  "contracts": 1,  "type": "PUT"  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "SHORT",  "close\_price": 1.23,  "contracts": 1,  "type": "CAll"  }  ]  }” | assert 200 == 200  (Objeto atualizado) |
| CC6 | **1-2-3-4-6-10-14-16-17-18** | rawPath = “/v1/share”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  “shared”: True  }” | assert 200 == 200  (Objeto compartilhado) |
| CC7 | **1-2-3-4-7-11-14-16-17-18** | rawPath = “/v1/delete”  body = “{  "id": "18199ad0-47fb-4592-8fb2-008726efab95",  “deleted”: True  }” | assert 200 == 200  (Objeto removido) |
| CC8 | **1-2-13** | rawPath = “/v1/invalid” | assert 500 == 500  (Erro – Rota não encontrada) |
| CC9 | **1-2-3-4-6-10-15** | rawPath = “/v1/share”  body = “{  "id": "1234"  }” | assert 500 = 500  (Entrada inválida) |
|  |  |  |  |

Tabela 10 - Teste em todos os caminhos para atualização de estratégia

# Teste de cadastro de usuário

O cadastro de usuário conta com os campos de email, nome de usuário e de senha. Todos os campos devem ser válidos antes da função de submit, que comunica com a API seja chamada. A seguir cada um dos campos passará pelos testes estruturais.

## Teste do Campo de Email

Para o campo do email ser válido ele deve atender os seguintes requisitos:

* Não ser vazio;
* Possuir um “@”;
* Possuir um ou mais “.”;
* Possuir texto depois do “.”.

### Função de validação do email

A seguir será apresentado a Função que irá validar o email:

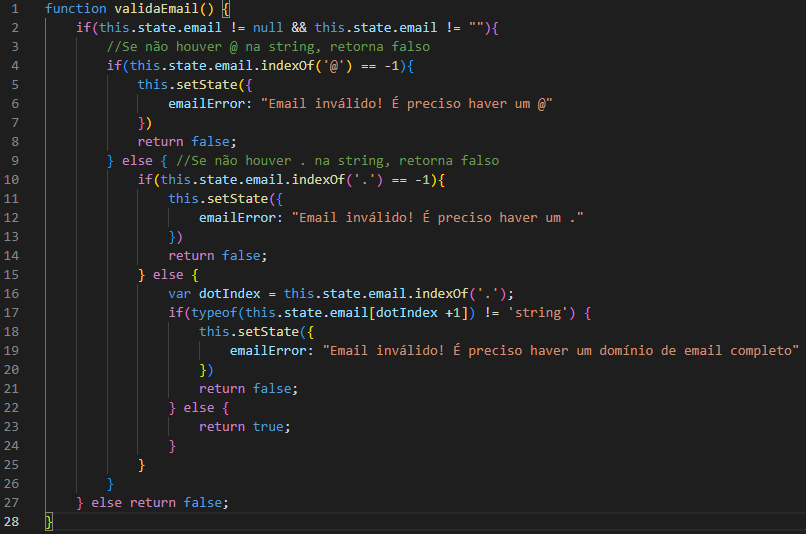


Figura 5 - Função de validação do email

### GFC para validação do Email

O Grafo de Fluxo de Controle segue numeração conforme as linhas de código da figura 3:

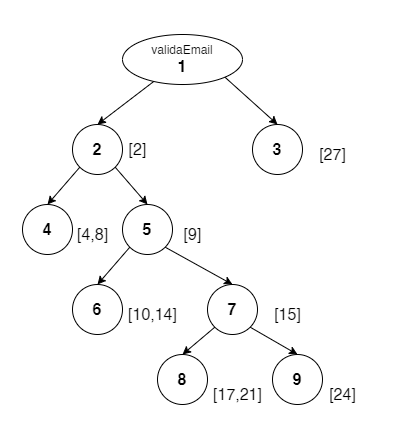


Figura 6 - GFC para validação do Email

### Análise de todos os nós do GFC para validação do Email

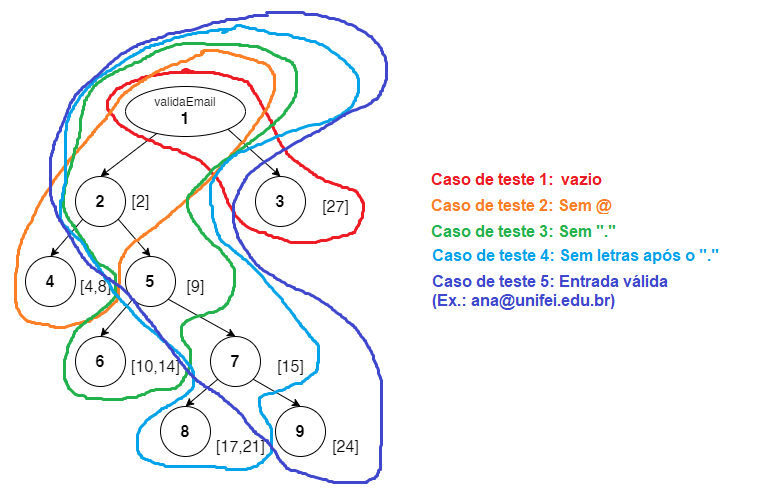


Figura 7 - Análise de todos os nós para validar email

### Análise de todas as arestas para validar email

Seguindo a imagem do GFC apresentada na figura 4, temos:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **ARESTAS** | **CASO/ENTRADA** | **SAIDA** |
| CT1 | **1-3** | Email = “” (vazio) | false |
| CT2 | **1-2, 2-4** | Email = “abc” | emailError = "Email inválido! É preciso haver um @"  false |
| CT3 | **1-2, 2-5, 5-6** | Email = “abc@unifei” | emailError = "Email inválido! É preciso haver um .”  false |
| CT4 | **1-2, 2-5, 5-7, 7-8** | Email = “abc@unifei.” | emailError = "Email inválido! É preciso haver um domínio de email completo”  false |
| CT5 | **1-2, 2-5, 5-7, 7-9** | Email = “abc@unifei.com” | true |

Tabela 11 – Teste para todas as arestas para validar email

## Teste do Campo de nome de usuário

Para o campo do nome do usuário ser válido ele deve atender os seguintes requisitos:

* Não ser vazio;
* Deve ter mais de 3 letras;
* Deve ter no mínimo 2 palavras;

### Função de validação do nome do usuário

A seguir será apresentado a Função que irá validar o nome do usuário:

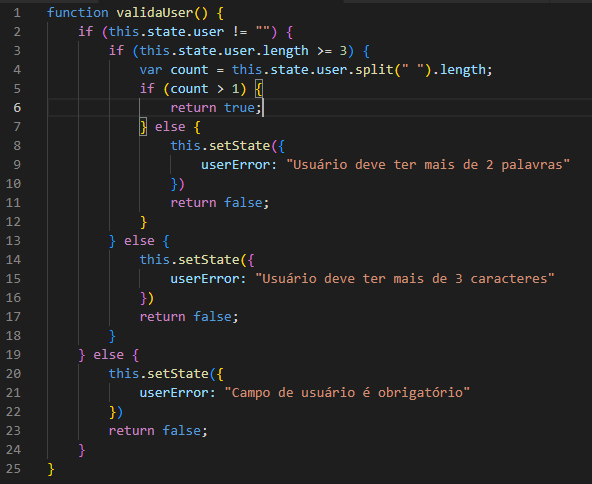


Figura 8 - Função de validação do nome do usuário

### FGC para validação do nome do usuário

O Grafo de Fluxo de Controle segue numeração conforme as linhas de código da imagem apresentada na figura 6, temos:

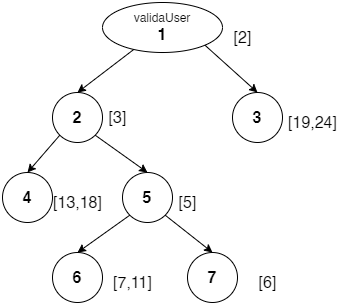


Figura 9 - FGC para validação do nome do usuário

### Análise de todos os nós para nome do usuário

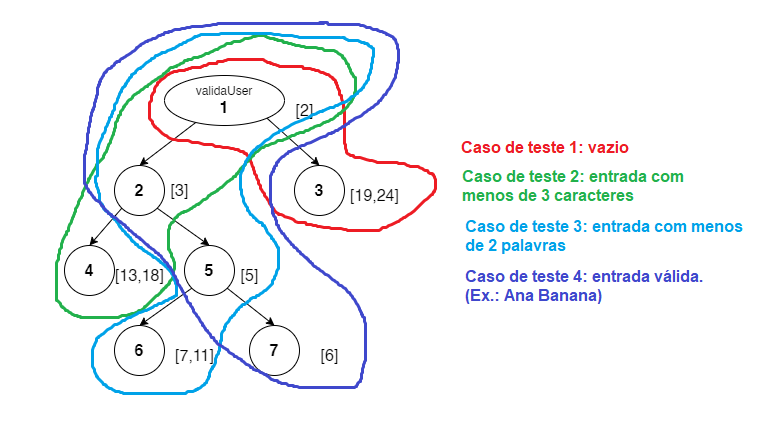


Figura 10 - Análise de todos os nós para nome do usuário

### Análise de todas as arestas para nome do usuário

Seguindo a imagem do GFC apresentada na figura 7, temos:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **ARESTAS** | **ENTRADA** | **SAIDA** |
| CT1 | **1-3** | Nome = “” (vazio) | userError = "Campo de usuário é obrigatório"  false |
| CT2 | **1-2, 2-4** | Nome = “Zé” | userError = Usuário deve ter mais de 3 caracteres"  false |
| CT3 | **1-2, 2-5, 5-6** | Nome = “Carlos” | userError = "Usuário deve ter mais de 2 palavras"  false |
| CT4 | **1-2, 2-5, 5-7** | Nome = “Zé Carlos” | true |

Tabela 12 - Análise de todas as arestas para nome do usuário

Teste

# Teste de calculo de payoff

O cálculo de payoff para estratégias é diferenciado a partir de campos type e transaction\_type. O Cálculo será então realizado em cima do exercise\_price e close\_price, considerando que o retorno é 200 para o teste passar e 500 em erro interno.

Os campos e seus possíveis valores, que influenciam nos cálculos são apresentados a seguir:

* type: “CALL” | “PUT”;
* transaction\_type: “SHORT” | “LONG”;
* exercise\_price: Decimal;
* close\_price: Decimal;

## Função de cálculo de Payoff

A seguir será apresentado a Função que irá validar o cálculo de Payoff:

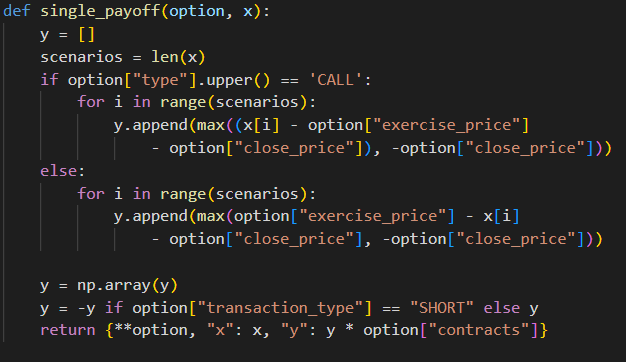


Figura 11 - Função de cálculo de Payoff

## GFC para Calculo de Payoff

O GFC segue numeração conforme a chamada da lambda\_handler abaixo:

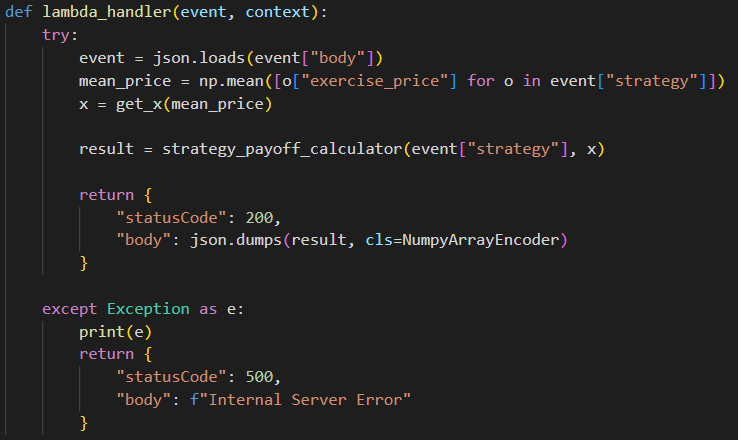


Figura 12 - Função lambda\_handler

A seguir é apresentado o GFC para cálculo de payoff:

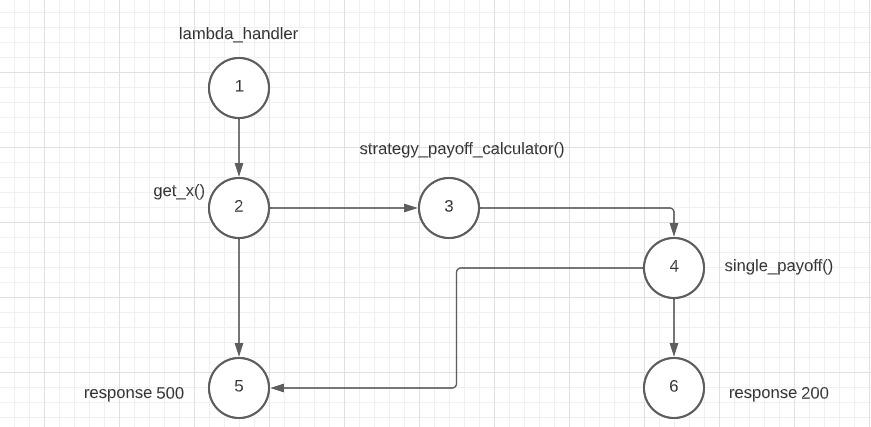


Figura 13 - GFC para Cálculo de Payoff

## Análise de todos os nós para validar o cálculo de Payoff

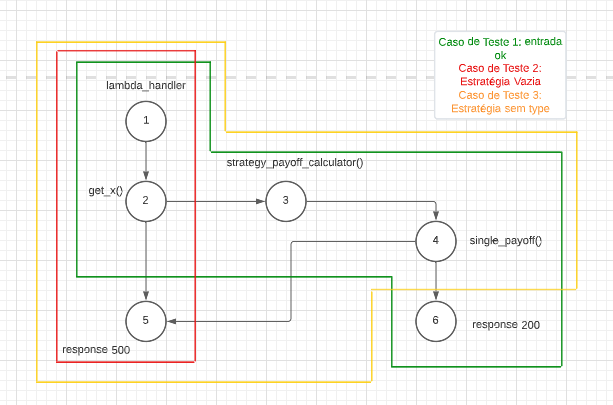


Figura 14 - Análise de todos os nós para validar o cálculo de Payoff

## Análise de todas as arestas para validar o cálculo de Payoff

Seguindo a imagem do GFC apresentada na figura 17, temos:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **ARESTAS** | **CASO/ENTRADA** | **SAIDA** |
| CT1 | **1-2,2-3,3-4,4-6** | base\_case = {  'body' : json.dumps({  "username": "blazzi",  "name": "Custom Long Straddle",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG",  "close\_price": 1.23,  "contracts": 1,  "type": "PUT"  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG",  "close\_price": 1.23,  "contracts": 1,  "type": "CALL"  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "SHORT",  "close\_price": 1.23,  "contracts": 1,  "type": "PUT"  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "SHORT",  "close\_price": 1.23,  "contracts": 1,  "type": "CALL"  }  ]  })  } | {  "statusCode": 200,  "body": json.dumps(result, cls=NumpyArrayEncoder)  } |
| CT2 | **1-2,2-5** | base\_case: {} | {  "statusCode": 500,  "body": f"Internal Server Error"  } |
| CT3 | **1-2, 2-3,3-4,4-5** | base\_case = {  'body' : json.dumps({  "username": "blazzi",  "name": "Custom Long Straddle",  "strategy": [  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG",  "close\_price": 1.23,  "contracts": 1  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "LONG",  "close\_price": 1.23,  "contracts": 1,  "type": "CALL"  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "SHORT",  "close\_price": 1.23,  "contracts": 1,  "type": "PUT"  },  {  "name": "custom",  "exercise\_price": 12.32,  "transaction\_type": "SHORT",  "close\_price": 1.23,  "contracts": 1,  "type": "CALL"  }  ]  })  } | {  "statusCode": 500,  "body": f"Internal Server Error"  } |

Tabela 13 - Teste de todas as arestas para validar o cálculo de Payoff

# Teste de Autenticação de usuário

A tela de autenticação do usuário conta com 2 campos para inserir as informações, sendo eles o campo de Email e o campo de Senha, os dois campos devem ser validados.

## Validação do campo de email

Para que um email seja considerado válido, deve:

* ter pelo menos uma ocorrência de:
  + caractere especial @ obrigatoriamente
  + letra [a-z,A-Z]
* Deve possuir um registro no sistema
* No mínimo 3 e no máximo 10 caracteres antes do @; de A - Z ou de 0 - 9 e ( \_ ) e ( . )
* No mínimo 3 e no máximo 10 caracteres depois do @; de A - Z ou de 0 - 9 e ( \_ ) e ( . )

### Código para a validação do campo de Email

O trecho de código abaixo é responsável por garantir a validade das informações inseridas no campo Email, seguindo os parametros estabelecidos no tópico 5.1.2

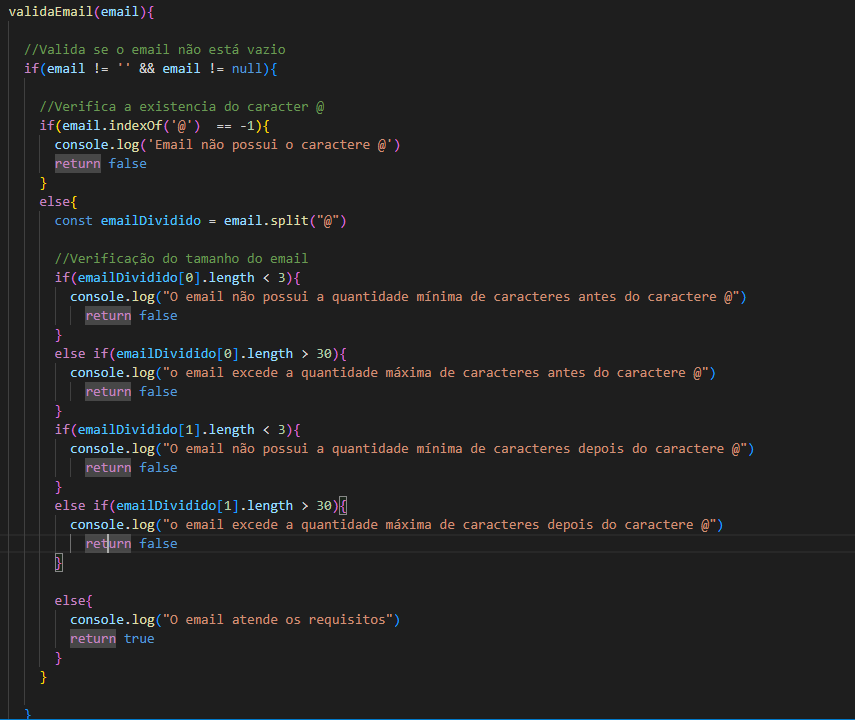


Figura 15 - Código para a validação do campo de Email

### FGC para validar campo de email

O Grafo de Fluxo de Controle do campo email segue numeração conforme as linhas de código da imagem apresentada no tópico 5.1.2.

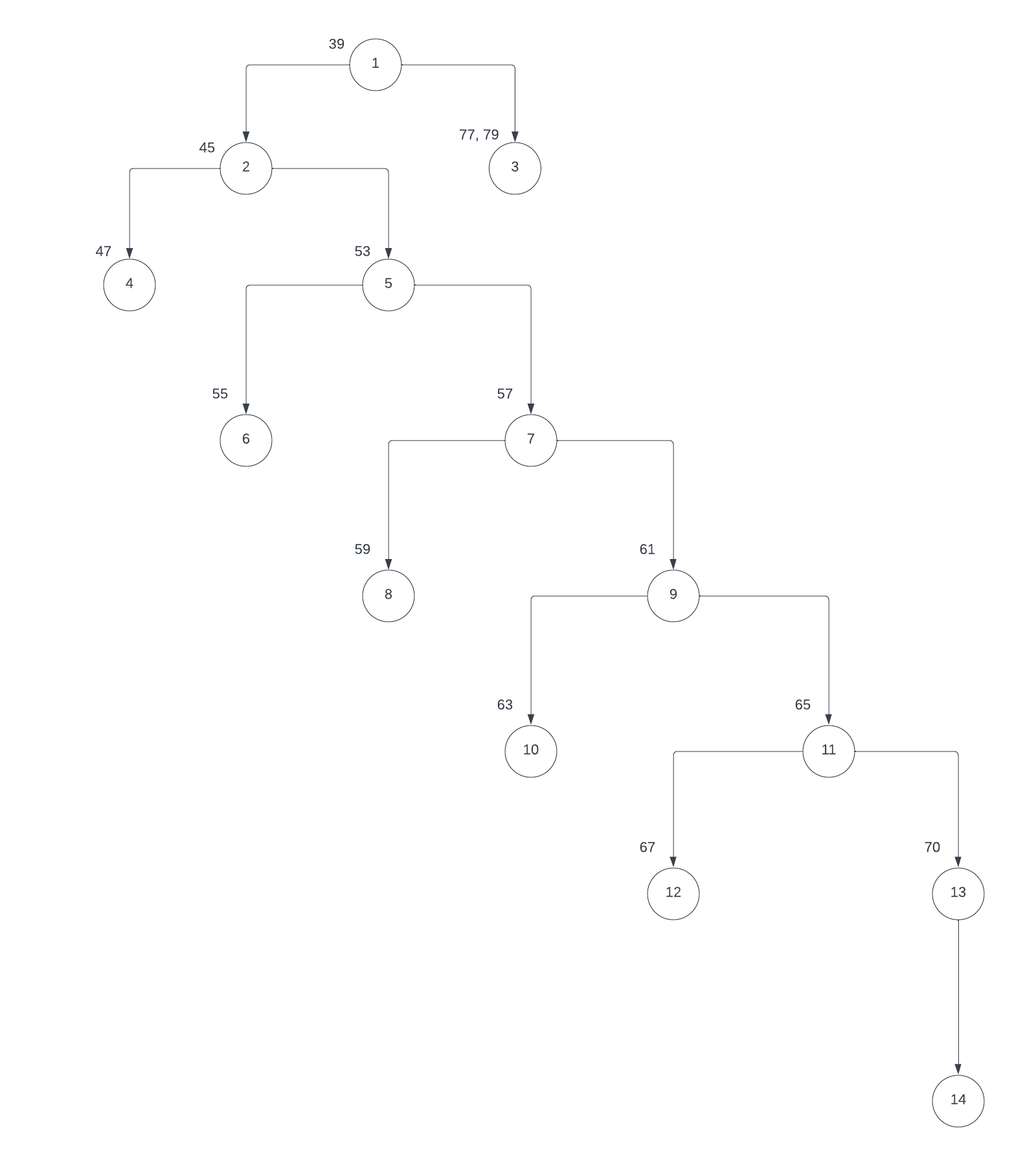


Figura 16- FGC para validar campo de email

### Análise de todos os nós para validar campo Email

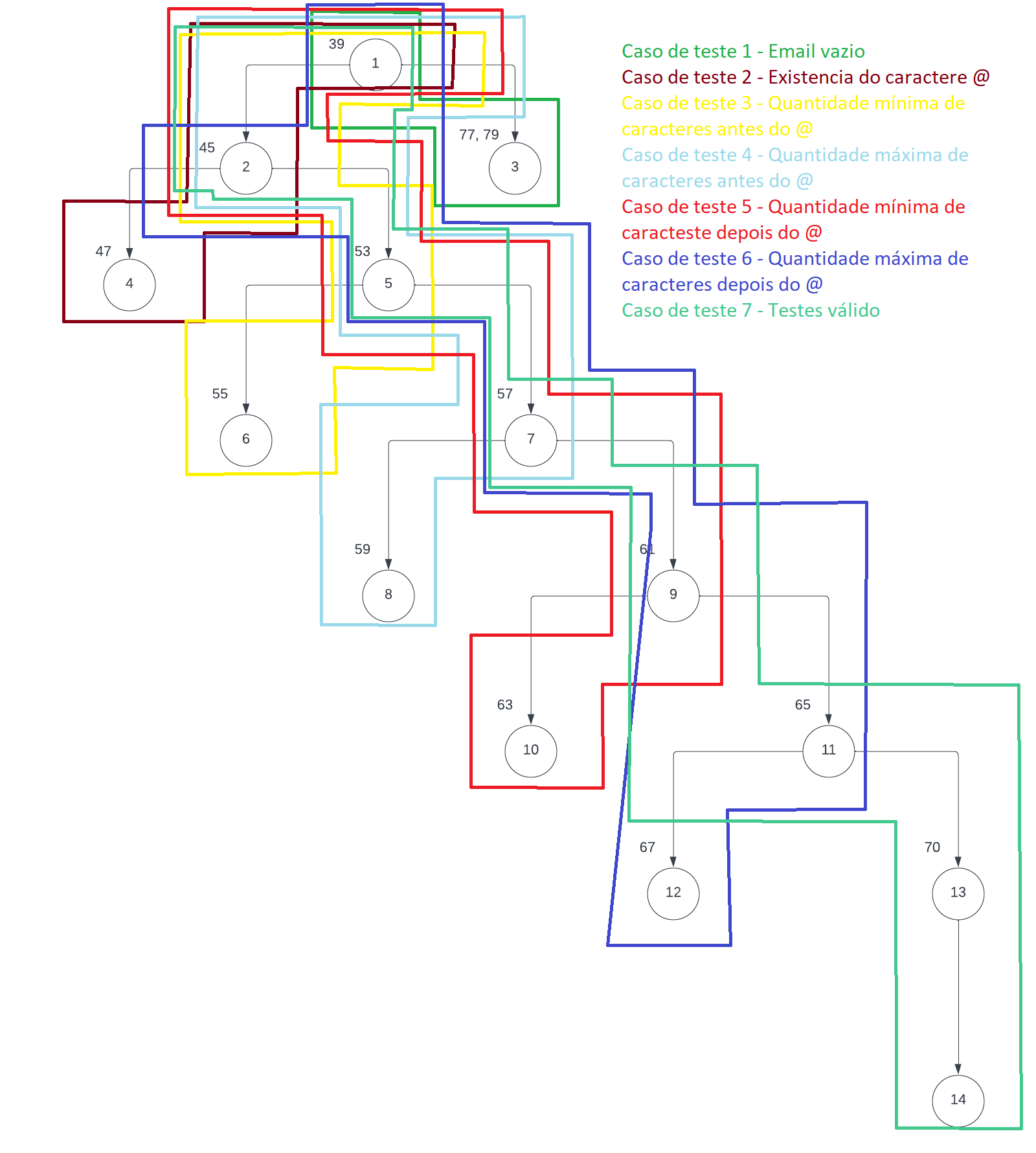


Figura 17 - Análise de todos os nós para validar campo Email

### Análise de todas as arestas para validar campo de email

Seguindo a imagem do GFC apresentada na figura 21, temos:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **ARESTAS** | **CASO/ENTRADA** | **SAIDA** |
| CT1 | **1-3** | Email = “” (vazio) | Campo email é obrigatório  false |
| CT2 | **1-2, 2-4** | Email = “Thiagohotmail.com” | É preciso pelo menos um caracter ‘@’  false |
| CT3 | **1-2, 2-5, 5-6** | Email = “th@hotmail.com” | É preciso pelo menos 3 caracteres antes do caractere ‘@’  false |
| CT4 | **1-2, 2-5, 5-7, 7-8** | Email = “thiagomarcelopassosunifei@hotmail.com” | É preciso ter menos de 20 caracteres antes do caractere ‘@’  false |
| CT5 | **1-2, 2-5, 5-7, 7-9, 9-10** | Email = thiagopass@ho | É preciso ter pelo menos 3 caractere após o caracter ‘@’ |
| CT6 | **1-2, 2-5, 5-7, 7-9, 9-11, 11 -12** | Email = thiagopass@universidadefederaldeitajubá@hotmail.com | É preciso ter menos de 20 caracteres depois do caractere ‘@’  false |
| CT7 | **1-2, 2-5, 5-7, 7-9, 9-11, 11 -13** | Email = d2018002850@unifei.edu.br | true |

Tabela 14 - Análise de todas as arestas para validar campo de email

## Validação do campo de Senha

* ter entre 8 e 20 caracteres,
* ter pelo menos uma ocorrência de:
  + caractere especial, por exemplo: [@, #, %, &, !,+]
  + número
  + letra [a-z,A-Z]
* Não conter o nome ou o ano de nascimento do usuário
* Deve possuir um registro no sistema

### Código para a validação do campo de Senha

O trecho de código abaixo é responsável por garantir a validade das informações inseridas no campo Senha, seguindo os parametros estabelecidos no tópico 5.1.4

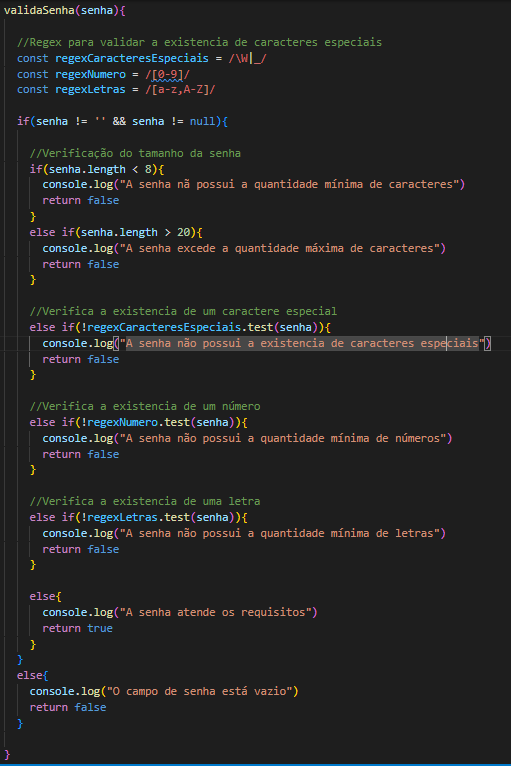


Figura 18 - Código para a validação do campo de Senha

### GFC do campo de senha

O Grafo de Fluxo de Controle do campo email segue numeração conforme as linhas de código da imagem apresentada no tópico 5.1.5.

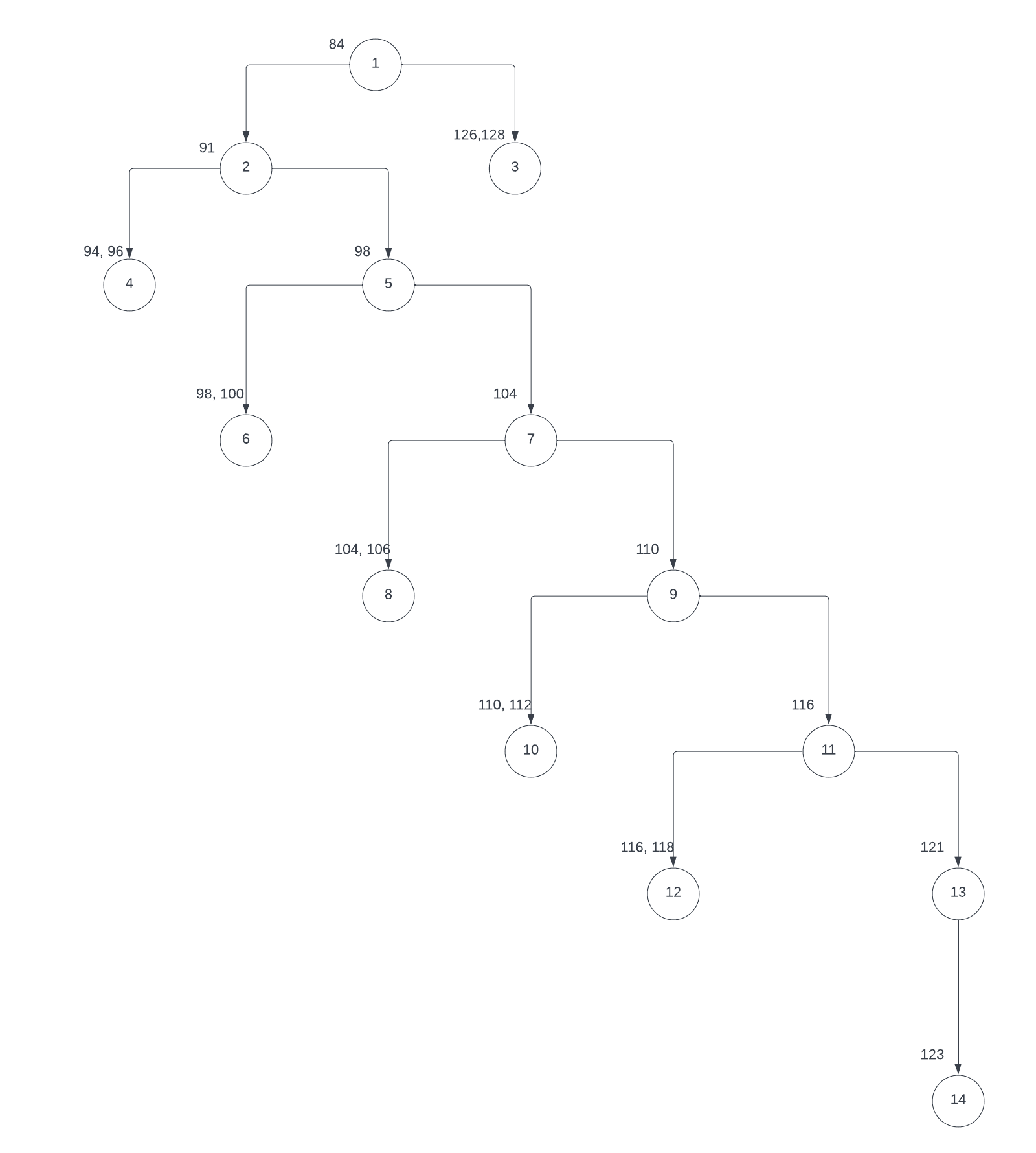


Figura 19 - GFC do campo de senha

### Análise de todos os nós para o campo Senha

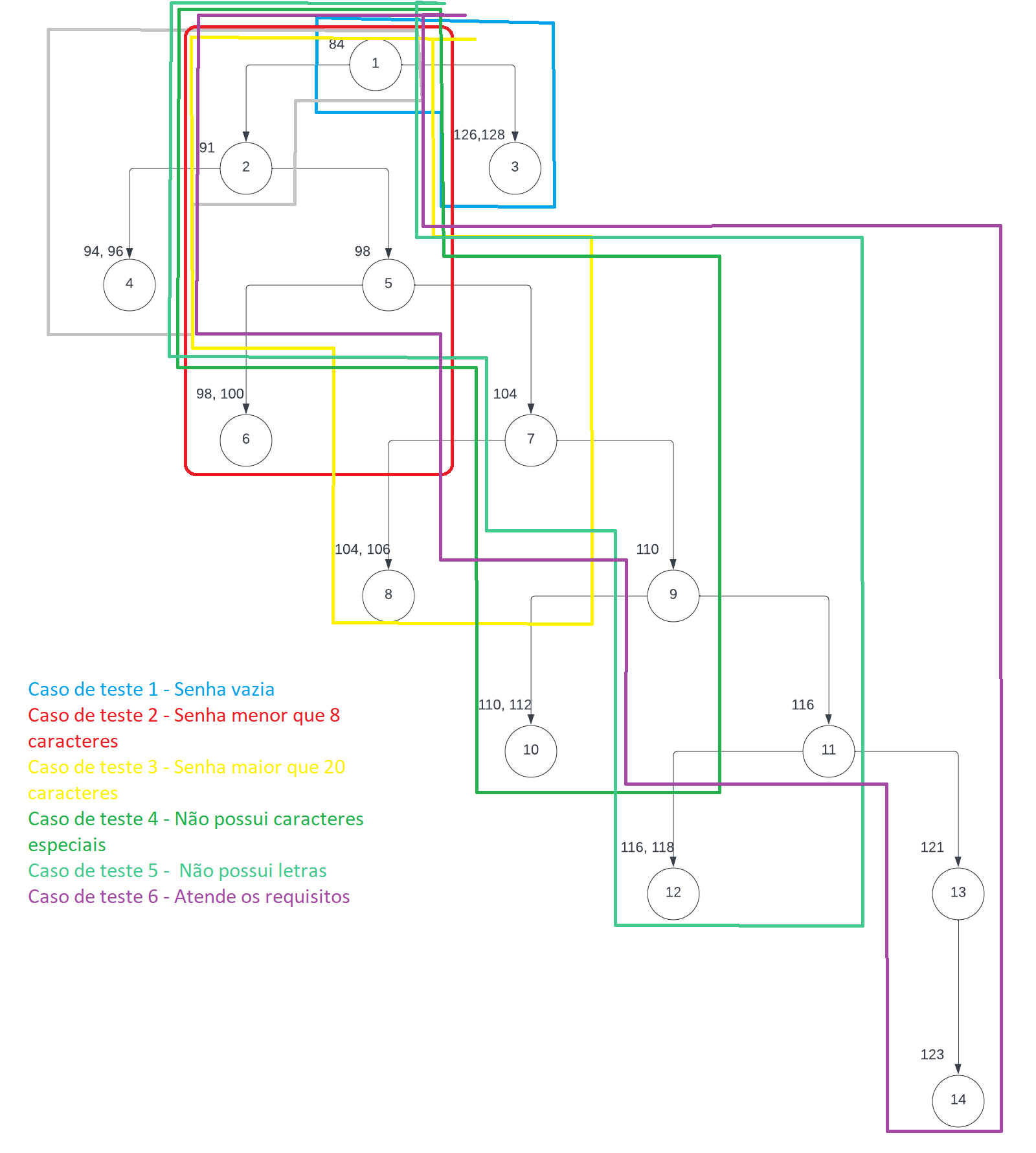


Figura 20 - Análise de todos os nós para o campo Senha

### Análise de todas as arestas do GFC para o campo senha

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **ARESTAS** | **CASO/ENTRADA** | **SAIDA** |
| CT1 | **1-3** | Senha = “” (vazio) | Campo email é obrigatório  false |
| CT2 | **1-2, 2-4** | Senha = “1dsa@” | É preciso pelo menos 8 caracteres |
| CT3 | **1-2, 2-5, 5-6** | Senha = “d4s55243dw#$wsdf21dsf2s!@$dsewe2312$$” | A senha deve ter menos de 20 caracteres |
| CT4 | **1-2, 2-5, 5-7, 7-8** | Senha = “1232Tsh32ss” | A senha deve ter ao menos 1 caractere especial  false |
| CT5 | **1-2, 2-5, 5-7, 7-9, 9-10** | Senha = daewd4%%@dsds | A senha deve possuir ao menos 1 número  false |
| CT6 | **1-2, 2-5, 5-7, 7-9, 9-11, 11 -12** | Senha = “23232@##@233” | A senha deve possuir ao menos uma letra  false |
| CT7 | **1-2, 2-5, 5-7, 7-9, 9-11, 11 -13** | Senha = “1ds@asTq$d” | true |

Tabela 15 - Análise de todas as arestas do GFC para o campo senha

# Teste da busca de opções Frontend

A tela de busca de opções conta com um campo para digitar o nome da opção a ser buscada. Para ser considerada um nome de opção válido deve conter:

* Começar com uma letra
* Os 3 próximos caracteres podem ser letras ou números
* O 5º caractere deve ser uma letra
* Os 3 últimos caracteres devem ser números
* Ter no máximo 8 caracteres

## Funções de controle da validação de busca de opções frontend

O grafo de fluxo de controle segue as seguintes funções para a validação de busca:

* g1: first\_letter → Verifica se começa com uma letra
* g2: name\_check → Verifica se os 3 próximos caracteres são letras ou números
* g3: type\_check → Verifica se o 5º caractere é uma letra
* g4: last\_numbers → Verifica se os 3 últimos caracteres são números
* g5: max\_lenght → Verifica se tem no máximo 8 caracteres para a busca
* g6: valid: estado final onde o nome da opção é válido
* g7: invalid: estado final onde o nome da opção é inválido

## GFC para busca de opções frontend

O Grafo de Fluxo de Controle do da busca de opções segue numeração conforme as funções apresentadas no tópico 7.1:

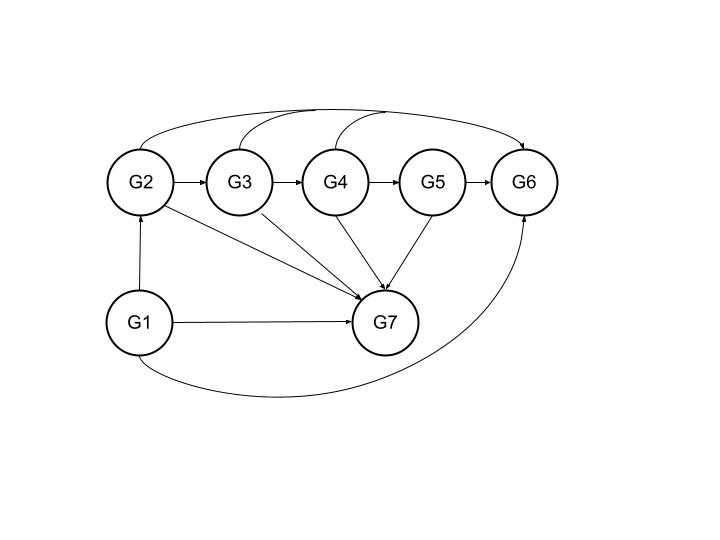


Figura 21 - GFC para busca de opções

## Análise de todos os nós para busca de opções frontend

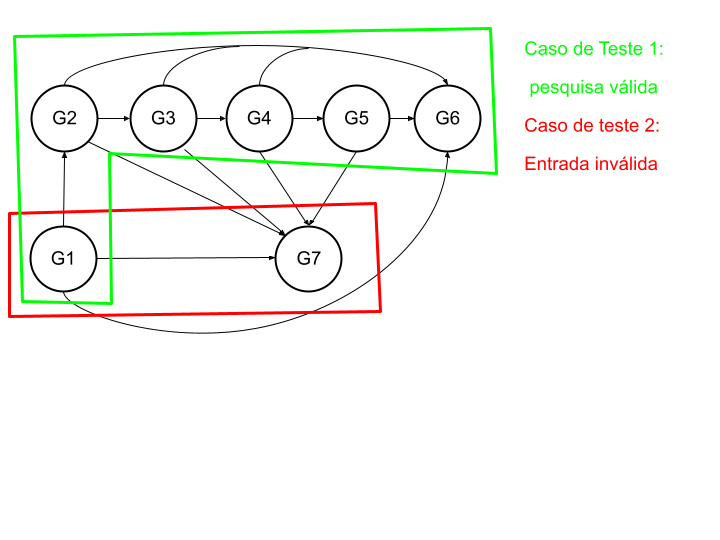


Figura 22 - Análise de todos os nós para busca de opções

## Análise de todas as arestas no GFC para busca de opções frontend

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **ARESTAS** | **CASO/ENTRADA** | **SAIDA** |
| CT1 | **1-7** | Busca = “33SAA101” | false, inicia com um número |
| CT2 | **1-6** | Busca = “B” | true, inicia com uma letra |
| CT3 | **1-2-7** | Busca = “B@” | false, 3 próximos caracteres não são letras ou números |
| CT4 | **1-2-6** | Busca = “B3SA” | true, 3 próximos caracteres são letras ou números |
| CT5 | **1-2-3-6** | Busca = “B3SAA” | true, 5º carater é uma letra |
| CT6 | **1-2-3-4-6** | Busca = “B3SAA1” | true, ultimos 3 caracteres são números |
| CT7 | **1-2-3-4-5-6** | Busca = “B3SAA1018” | false, maior que 8 caracteres |

Tabela 16 - Análise de todas as arestas no GFC para busca de opções frontend

# Teste da busca de opções backend

A Busca de opções é utilizada para buscar informações da opção a fim de mostrar ao usuário tais dados requeridos.

Informações relevantes para o teste:

* A lambda\_handler recebe event e context, nesse caso não sendo utilizados.
* O event deve possuir o id e name da opção desejada, por onde a busca será feita.
* O retorno deve ser com status code 200 + informações da opção para o teste passar e 404/Exception em caso de não encontrada/envio do event inválido.

Os campos e seus possíveis valores, que influenciam nos cálculos são apresentados a seguir:

* name: String
* id: String

## Função de busca de opção backend

Segue a função de busca de opção

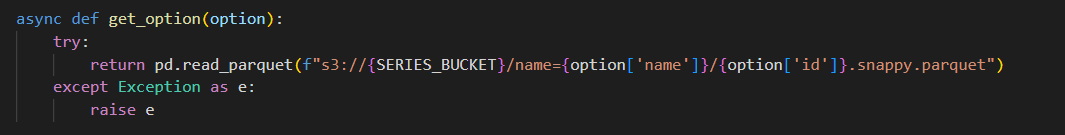


Figura 23 - Busca de opção (backend)

## Análise de todos os nós para busca de opção backend

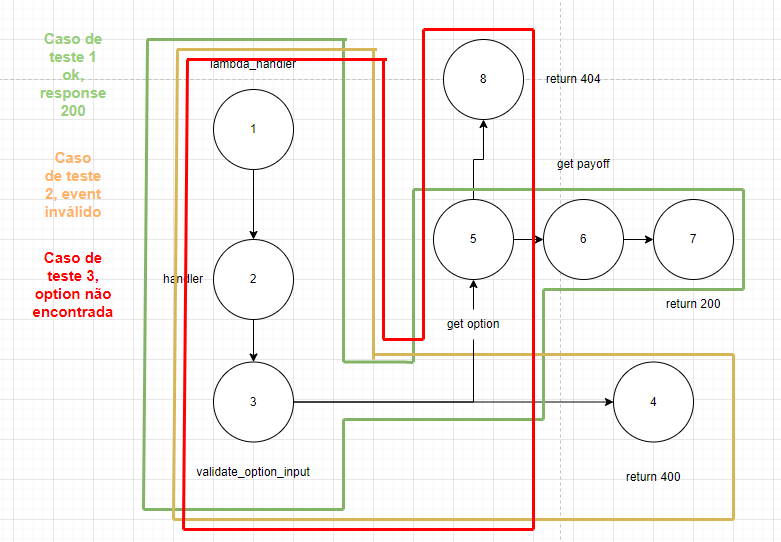


Figura 24 - Análise de todos os nós para busca de opção backend

## Análise de todas as arestas para validar o cálculo de payoff backend

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Arestas** | **Caso/Entrada** | **Saída** |
| CT1 | 1-2,2-3,3-5,5-6,6-7 | base\_case = {          "event": {              "body": '{"id": "60137665a3315885b579abe6803b55d0", "name": "BOVAA100"}'          },          "context": None      } | {    "statusCode": 200,    "body": json.dumps(result)  } |
| CT2 | 1-2,2-3,3-4 | base\_case={} | {    "statusCode": 400,    "body": f"bad request, body not found at event"  } |
| CT3 | 1-2,2-3,3-5,5-8 | base\_case={    "event": {      "body": '{"id": "ABCD7665a3315885b579abe6803b55d0", "name": "BOVAA100"}'    },    "context": None  }  **or**  base\_case={    "event": {      "body": '{"id": "60137665a3315885b579abe6803b55d0", "name": "ABCDEFGH"}'    },    "context": None  } | {    "statusCode": 404,    "body": f"'Option not found!"  } |

Tabela 17 - Análise de todas as arestas para validar o cálculo de payoff backend

## Teste Search

Utilizado para busca de opções.

Informações relevantes para o teste:

* A lambda\_handler recebe event e context, nesse caso não sendo utilizados.
* Os dados estão mockados para que não consuma recursos da AWS no momento, então o retorno sempre deve ser status code 200 + list, caso a cloud function esteja up

## Função de Search

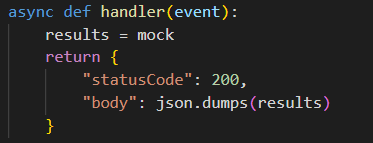


Figura 25 - Função de Search

## **Análise de todos os nós para** Search

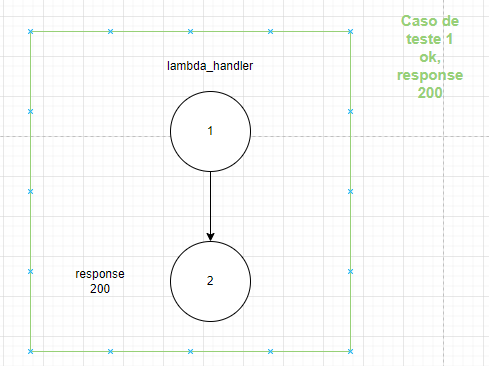


Figura 26 - Análise de todos os nós para Search

## Análise de todas as arestas para validar o cálculo de payoff do Backend

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Arestas | Caso/Entrada | Saída |
| CT1 | 1-2 | None | {          "statusCode": 200,          "body": json.dumps(list)   } |

Figura 27 - Análise de todas as arestas para validar o cálculo de payoff do Backend

# Teste de health ckeck

Utilizado para pingar na cloud functions, indica se o servidor está ou não up. O retorno é 200 para caso o teste passe e timeout caso contrário.

A lambda\_handler recebe event e context, nesse caso não sendo utilizados

## Função Health

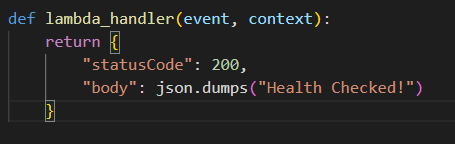


Figura 28 - Função Health

## GFC para cálculo de health check

 seguir segue o GFC para cálculo de health check. Ele apenas responde caso o recurso esteja up, não fazendo parte do seu sistema a não resposta.

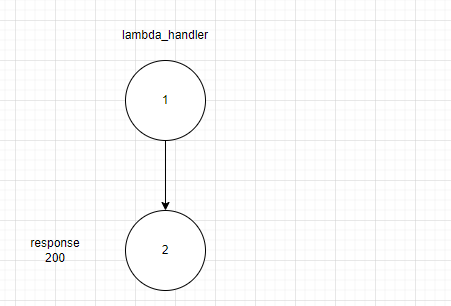


Figura 29 - GFC para cálculo de health check

## Análise de todos os nós para validar o health check

A lambda\_handler, ao receber uma requisição, retorna um json com status code 200, e a mensagem informando que o serviço se encontra disponível.

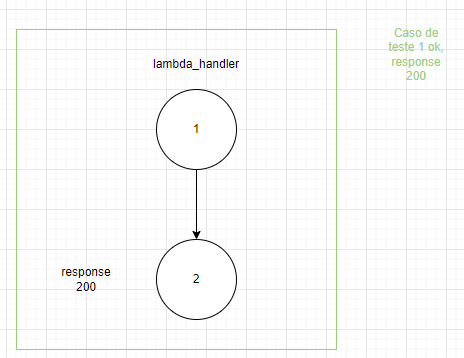


Figura 30 - Análise de todos os nós para validar o health check

## Análise de todas as arestas para validar o health

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Arestas | Caso/Entrada | Saída |
| CT1 | 1-2 | None | {          "statusCode": 200,          "body": json.dumps("Health Checked!")      } |

Tabela 18 - Análise de todas as arestas para validar o health

# Teste de inserção de opção fictícia

O cadastro de opção fictícia exige inserir um nome, escolher o tipo da operação e da transação, inserir o número de contratos, o preço da opção e o preço underlying. Todos os campos devem ser válidos para que a opção fictícia seja inserida.

## Teste de validade de inserção

Para que a opção fictícia seja inserida, os requisitos de cada campo (requisitos descritos nos testes combinatórios - item 8 do documento anexo “DTF Caixa-Preta”) têm que ser atendidos. A verificação de validade dos dados é definida pela função a seguir:

### Função de validação de inserção

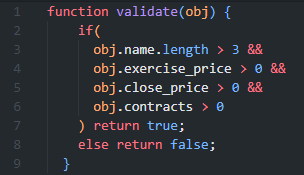


Figura - Função de validação da inserção

### GFC para validação de inserção

O Grafo de Fluxo de Controle segue numeração conforme as linhas de código da figura 3:

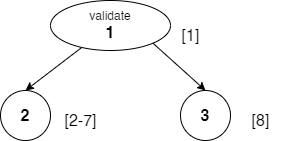


Figura 32 – CFG de validação de inserção

### Análise de todos os nós do GFC para validação de inserção

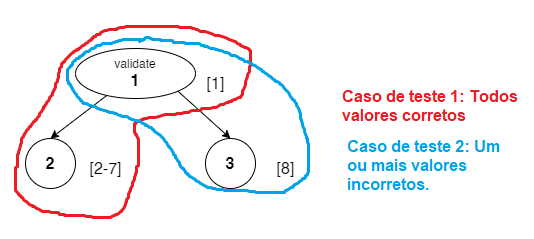


Figura 33 – Análise de todos os nós para validar inserção

### Análise de todas as arestas para validar inserção

Seguindo a imagem do GFC apresentada na figura 4, temos:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **ARESTAS** | **CASO/ENTRADA** | **SAIDA** |
| CT1 | **1-2** | Obj = {“name”: “ABCD”, “exercise\_price”: 3, “close\_price”: 7.5, “contracts”: 2} | true |
| CT2 | **1-3** | Obj = {“name”: “A”, “exercise\_price”: -3, “close\_price”: 7.5, “contracts”: 2} | false |

Tabela 19 - Teste para todas as arestas para validar inserção