

Underlying

**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**



**UNIFEI**  
Universidade Federal de Itajubá

**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)



Underlying



## Revisões do Documento

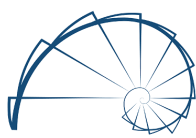
Revisões são melhoramentos na estrutura do documento e também no seu conteúdo. O objetivo primário desta tabela é a fácil identificação da versão do documento. Toda modificação no documento deve constar nesta tabela.

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

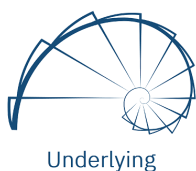


Underlying



## Índice de Ilustrações

FIGURA 1 - COBERTURA DE TESTES – SERVIÇO DE ESTRATÉGIAS .....	11
FIGURA 2 - COBERTURA DE TESTES – SERVIÇO DE OPÇÕES .....	12
FIGURA 3 - TESTE PARA VISUALIZAÇÃO DAS ESTRATÉGIAS .....	17
FIGURA 4 - GFC PARA EDIÇÃO DA ESTRATÉGIA .....	20
FIGURA 5 - FUNÇÃO DE VALIDAÇÃO DO EMAIL .....	40
FIGURA 6 - GFC PARA VALIDAÇÃO DO EMAIL .....	40
FIGURA 7 - ANÁLISE DE TODOS OS NÓS PARA VALIDAR EMAIL .....	41
FIGURA 8 - FUNÇÃO DE VALIDAÇÃO DO NOME DO USUÁRIO .....	42
FIGURA 9 - FGC PARA VALIDAÇÃO DO NOME DO USUÁRIO .....	43
FIGURA 10 - ANÁLISE DE TODOS OS NÓS PARA NOME DO USUÁRIO .....	43
FIGURA 11 - FUNÇÃO DE CÁLCULO DE PAYOFF .....	45
FIGURA 12 - FUNÇÃO LAMBDA_HANDLER .....	45
FIGURA 13 - GFC PARA CÁLCULO DE PAYOFF .....	46
FIGURA 14 - ANÁLISE DE TODOS OS NÓS PARA VALIDAR O CÁLCULO DE PAYOFF .....	46
FIGURA 15 - CÓDIGO PARA A VALIDAÇÃO DO CAMPO DE EMAIL .....	50
FIGURA 16 - FGC PARA VALIDAR CAMPO DE EMAIL .....	51
FIGURA 17 - ANÁLISE DE TODOS OS NÓS PARA VALIDAR CAMPO EMAIL .....	52
FIGURA 18 - CÓDIGO PARA A VALIDAÇÃO DO CAMPO DE SENHA .....	54
FIGURA 19 - GFC DO CAMPO DE SENHA .....	55
FIGURA 20 - ANÁLISE DE TODOS OS NÓS PARA O CAMPO SENHA .....	56
FIGURA 21 - GFC PARA BUSCA DE OPÇÕES .....	58
FIGURA 22 - ANÁLISE DE TODOS OS NÓS PARA BUSCA DE OPÇÕES .....	58
FIGURA 23 - BUSCA DE OPÇÃO (BACKEND) .....	59
FIGURA 24 - ANÁLISE DE TODOS OS NÓS PARA BUSCA DE OPÇÃO BACKEND .....	60
FIGURA 25 - FUNÇÃO DE SEARCH .....	62
FIGURA 26 - ANÁLISE DE TODOS OS NÓS PARA SEARCH .....	62
FIGURA 27 - ANÁLISE DE TODAS AS ARESTAS PARA VALIDAR O CÁLCULO DE PAYOFF DO BACKEND .....	62
FIGURA 28 - FUNÇÃO HEALTH .....	63
FIGURA 29 - GFC PARA CÁLCULO DE HEALTH CHECK .....	63
FIGURA 30 - ANÁLISE DE TODOS OS NÓS PARA VALIDAR O HEALTH CHECK .....	64
FIGURA 31 - FUNÇÃO DE VALIDAÇÃO DA INSERÇÃO .....	65
FIGURA 32 - CFG DE VALIDAÇÃO DE INSERÇÃO .....	65
FIGURA 33 - ANÁLISE DE TODOS OS NÓS PARA VALIDAR INSERÇÃO .....	66

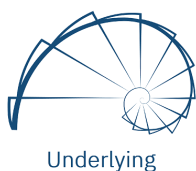


Underlying



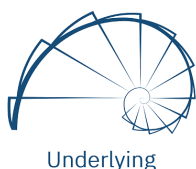
## Índice de Tabelas

TABELA 1 - RESPONSABILIDADE DOS TESTES ESTRUTURAIS	9
TABELA 2 - GFC PARA VISUALIZAÇÃO DA ESTRATÉGIA	13
TABELA 3 - ANÁLISE DO GFC PARA TESTE EM TODOS OS NÓS DA LEITURA DA ESTRATÉGIA	14
TABELA 4 - TESTE EM TODOS OS NÓS DA LEITURA DA ESTRATÉGIA	15
TABELA 5 - ANÁLISE DO GFC TESTE DE TODAS AS ARESTAS NA LEITURA DA ESTRATÉGIA	15
TABELA 6 - TESTE EM TODAS AS ARESTAS DA LEITURA DA ESTRATÉGIA	16
TABELA 7 - TESTE EM TODOS OS CAMINHOS DA VISUALIZAÇÃO DA ESTRATÉGIA	16
TABELA 8 - TESTE EM TODOS OS NÓS PARA ATUALIZAÇÃO DE ESTRATÉGIA	24
TABELA 9 - TESTE EM TODAS AS ARESTAS DA ATUALIZAÇÃO DA ESTRATÉGIA	31
TABELA 10 - TESTE EM TODOS OS CAMINHOS PARA ATUALIZAÇÃO DE ESTRATÉGIA	39
TABELA 11 - TESTE PARA TODAS AS ARESTAS PARA VALIDAR EMAIL	41
TABELA 12 - ANÁLISE DE TODAS AS ARESTAS PARA NOME DO USUÁRIO	44
TABELA 13 - TESTE DE TODAS AS ARESTAS PARA VALIDAR O CÁLCULO DE PAYOFF	49
TABELA 14 - ANÁLISE DE TODAS AS ARESTAS PARA VALIDAR CAMPO DE EMAIL	53
TABELA 15 - ANÁLISE DE TODAS AS ARESTAS DO GFC PARA O CAMPO SENHA	57
TABELA 16 - ANÁLISE DE TODAS AS ARESTAS NO GFC PARA BUSCA DE OPÇÕES FRONTEND	59
TABELA 17 - ANÁLISE DE TODAS AS ARESTAS PARA VALIDAR O CÁLCULO DE PAYOFF BACKEND	61
TABELA 18 - ANÁLISE DE TODAS AS ARESTAS PARA VALIDAR O HEALTH	64
TABELA 19 - TESTE PARA TODAS AS ARESTAS PARA VALIDAR INSERÇÃO	66

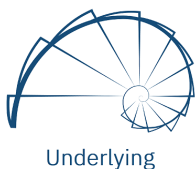


## Sumário

<b>1. Introdução .....</b>	<b>7</b>
1.1 Técnica do teste estrutural .....	7
1.2 Responsabilidades de testes da equipe de projeto .....	8
<b>2. COBERTURA DOS TESTES .....</b>	<b>9</b>
<b>3. Teste da visualização das estratégias .....</b>	<b>13</b>
3.1 Criação do GFC para visualização da estratégia.....	13
3.2 Teste Caixa Branca: Técnica do caminho para a leitura das estratégias.....	13
3.3 Teste em todos os nós para a leitura das estratégias .....	14
2.3. Teste em todas as arestas para visualizar estratégia .....	15
2.4. Teste em todos os caminhos para visualizar estratégia .....	16
<b>3. Teste da edição das estratégias .....</b>	<b>17</b>
3.1. Criação do GFC para edição da estratégia.....	18
3.2. Teste em todos os nós para atualização de estratégia.....	20
3.3. Teste em todas as arestas para atualização de estratégia .....	24
3.4. Teste em todos os caminhos para atualização de estratégia .....	31
<b>4. Teste de cadastro de usuário .....</b>	<b>39</b>
4.1. Teste do Campo de Email.....	39
4.1.1. Função de validação do email .....	39
4.1.2. GFC para validação do Email .....	40
4.1.3. Análise de todos os nós do GFC para validação do Email .....	41
4.1.4. Análise de todas as arestas para validar email.....	41
4.2. Teste do Campo de nome de usuário .....	42
4.2.1. Função de validação do nome do usuário.....	42
4.2.2. FGC para validação do nome do usuário.....	42
4.2.3. Análise de todos os nós para nome do usuário .....	43
4.2.4. Análise de todas as arestas para nome do usuário .....	43
<b>5. Teste de calculo de payoff .....</b>	<b>44</b>
5.1. Função de cálculo de Payoff.....	44
5.2. GFC para Calculo de Payoff .....	45
5.3. Análise de todos os nós para validar o cálculo de Payoff.....	46
5.4. Análise de todas as arestas para validar o cálculo de Payoff .....	47



<b>6. Teste de Autenticação de usuário .....</b>	<b>49</b>
<b>6.1. Validação do campo de email .....</b>	<b>50</b>
6.1.1. Código para a validação do campo de Email.....	50
6.1.2. FGC para validar campo de email.....	51
6.1.3. Análise de todos os nós para validar campo Email .....	52
6.1.4. Análise de todas as arestas para validar campo de email .....	53
<b>6.2. Validação do campo de Senha .....</b>	<b>53</b>
6.2.1. Código para a validação do campo de Senha.....	53
6.2.2. GFC do campo de senha.....	55
6.2.3. Análise de todos os nós para o campo Senha .....	56
6.2.4. Análise de todas as arestas do GFC para o campo senha .....	57
<b>7. Teste da busca de opções Frontend.....</b>	<b>57</b>
7.1. Funções de controle da validação de busca de opções frontend.....	57
7.2. GFC para busca de opções frontend .....	58
7.3. Análise de todos os nós para busca de opções frontend .....	58
7.4. Análise de todas as arestas no GFC para busca de opções frontend.....	59
<b>8. Teste da busca de opções backend.....</b>	<b>59</b>
8.1. Função de busca de opção backend .....	59
8.2. Análise de todos os nós para busca de opção backend .....	60
8.3. Análise de todas as arestas para validar o cálculo de payoff backend.....	60
8.4. Teste Search.....	61
8.5. Função de Search .....	61
8.6. Análise de todos os nós para Search .....	62
8.7. Análise de todas as arestas para validar o cálculo de payoff do Backend .....	62
<b>9. Teste de health check .....</b>	<b>63</b>
9.1. Função Health .....	63
9.2. GFC para cálculo de health check .....	63
9.3. Análise de todos os nós para validar o health check .....	64
9.4. Análise de todas as arestas para validar o health.....	64
<b>10. Teste de inserção de opção fictícia .....</b>	<b>64</b>
<b>10.1. Teste de validade de inserção .....</b>	<b>64</b>
10.1.1. Função de validação de inserção.....	65
10.1.2. GFC para validação de inserção.....	65
10.1.3. Análise de todos os nós do GFC para validação de inserção .....	66
10.1.4. Análise de todas as arestas para validar inserção .....	66



## 1. 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.

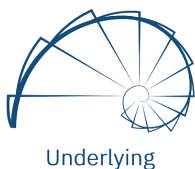
Essa técnica é vista como complementar à técnica funcional e informações obtidas pela aplicação desses critérios têm sido consideradas relevantes para as atividades de manutenção, depuração e para a confiabilidade de *software*. A técnica de teste de caixa-branca é recomendada para as fases de teste 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 e, portanto, contribuindo com uma maior qualidade e confiabilidade.

### 1.1 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. Esta 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, conseqüentemente, 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

## 1.2 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	Realiza teste para a edição dos dados de entrada referentes a	Lucas

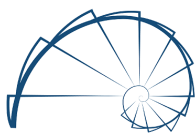


(backend)	uma estratégia de opções no banco de dados DynamoDB	
Leitura das estratégias de opções (backend)	Realiza o teste para leitura/visualização da estratégia de opções (backend) no banco DynamoDB	Ivan
Cálculo de Payoff (backend)	Realiza os testes para Cálculo de Payoff em Estratégias	Bruno Brandão 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 salvar 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**

## 2. 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



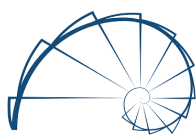
Underlying



(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:

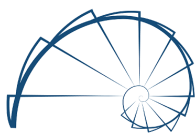


Underlying



Coverage report: 97%				
coverage.py v6.4.2, created at 2022-07-17 02:23 -0300				
Module	statements	missing	excluded	coverage
strategies/functions/api/create/__init__.py	0	0	0	100%
strategies/functions/api/create/app/__init__.py	0	0	0	100%
strategies/functions/api/create/app/app.py	31	15	0	52%
strategies/functions/api/create/app/schema.py	28	0	0	100%
strategies/functions/api/create/tests/__init__.py	0	0	0	100%
strategies/functions/api/create/tests/input.py	348	0	0	100%
strategies/functions/api/create/tests/test_lambda_handler.py	0	0	0	100%
strategies/functions/api/create/tests/test_validate_item.py	254	0	0	100%
strategies/functions/api/payoff/__init__.py	0	0	0	100%
strategies/functions/api/payoff/app/__init__.py	0	0	0	100%
strategies/functions/api/payoff/app/app.py	40	1	0	98%
strategies/functions/api/payoff/test/__init__.py	0	0	0	100%
strategies/functions/api/payoff/test/input.py	20	0	0	100%
strategies/functions/api/payoff/test/test_coverage.py	19	2	0	89%
strategies/functions/api/read/__init__.py	0	0	0	100%
strategies/functions/api/read/app/__init__.py	0	0	0	100%
strategies/functions/api/read/app/app.py	54	10	0	81%
strategies/functions/api/read/app/schema.py	28	2	0	93%
strategies/functions/api/read/tests/__init__.py	0	0	0	100%
strategies/functions/api/read/tests/input_coverage.py	16	0	0	100%
strategies/functions/api/read/tests/test_coverage.py	15	0	0	100%
strategies/functions/api/read/tests/testenv/__init__.py	0	0	0	100%
strategies/functions/api/read/tests/testenv/aws.py	29	2	0	93%
strategies/functions/api/read/tests/testenv/data.py	10	0	0	100%
strategies/functions/api/read/tests/testenv/resources.py	11	0	0	100%
strategies/functions/api/read/tests/testenv/test_resources.py	19	0	0	100%
strategies/functions/api/update/__init__.py	0	0	0	100%
strategies/functions/api/update/app/__init__.py	0	0	0	100%
strategies/functions/api/update/app/app.py	74	1	0	99%
strategies/functions/api/update/app/schema.py	28	2	0	93%
strategies/functions/api/update/tests/__init__.py	0	0	0	100%
strategies/functions/api/update/tests/input.py	176	0	0	100%
strategies/functions/api/update/tests/input_coverage.py	33	0	0	100%
strategies/functions/api/update/tests/test_coverage.py	25	0	0	100%
strategies/functions/api/update/tests/test_validate_delete.py	65	0	0	100%
strategies/functions/api/update/tests/test_validate_share.py	65	0	0	100%
strategies/functions/api/update/tests/testenv/__init__.py	0	0	0	100%
strategies/functions/api/update/tests/testenv/aws.py	29	2	0	93%
strategies/functions/api/update/tests/testenv/data.py	10	0	0	100%
strategies/functions/api/update/tests/testenv/resources.py	11	0	0	100%
strategies/functions/api/update/tests/testenv/test_resources.py	19	0	0	100%
Total	1457	37	0	97%

Figura 1 - Cobertura de testes – Serviço de Estratégias



Underlying

coverage.py v6.4.2, created at 2022-07-17 02:22 -0300

Module	statements	missing	excluded	coverage
options/functions/api/health/__init__.py	0	0	0	100%
options/functions/api/health/app/__init__.py	0	0	0	100%
options/functions/api/health/app/app.py	3	0	0	100%
options/functions/api/health/test/__init__.py	0	0	0	100%
options/functions/api/health/test/test_health.py	9	0	0	100%
options/functions/api/info/__init__.py	0	0	0	100%
options/functions/api/info/app/__init__.py	0	0	0	100%
options/functions/api/info/app/app.py	43	20	0	53%
options/functions/api/info/app/builder.py	34	23	0	32%
options/functions/api/info/app/schema.py	4	0	0	100%
options/functions/api/info/test/__init__.py	0	0	0	100%
options/functions/api/info/test/input.py	21	0	0	100%
options/functions/api/info/test/output.py	22	0	0	100%
options/functions/api/info/test/test_info.py	80	2	0	98%
options/functions/api/search/__init__.py	0	0	0	100%
options/functions/api/search/app/__init__.py	0	0	0	100%
options/functions/api/search/app/app.py	16	4	0	75%
options/functions/api/search/test/__init__.py	0	0	0	100%
options/functions/api/search/test/input.py	17	4	0	76%
options/functions/api/search/test/output.py	17	4	0	76%
options/functions/api/search/test/test_search.py	21	0	0	100%
options/functions/services/payoff/__init__.py	0	0	0	100%
options/functions/services/payoff/app/__init__.py	0	0	0	100%
options/functions/services/payoff/app/app.py	37	8	0	78%
options/functions/services/payoff/test/__init__.py	0	0	0	100%
options/functions/services/payoff/test/input.py	17	5	0	71%
options/functions/services/payoff/test/test_coverage_.py	11	2	0	82%
options/test_env/aws.py	35	10	0	71%
options/test_env/data.py	2	1	0	50%
options/test_env/resources.py	17	6	0	65%
options/test_env/test_resources.py	8	0	0	100%
Total	414	89	0	79%

**Figura 2 - Cobertura de testes – Serviço de Opções**

### 3. 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.

#### 3.1 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.

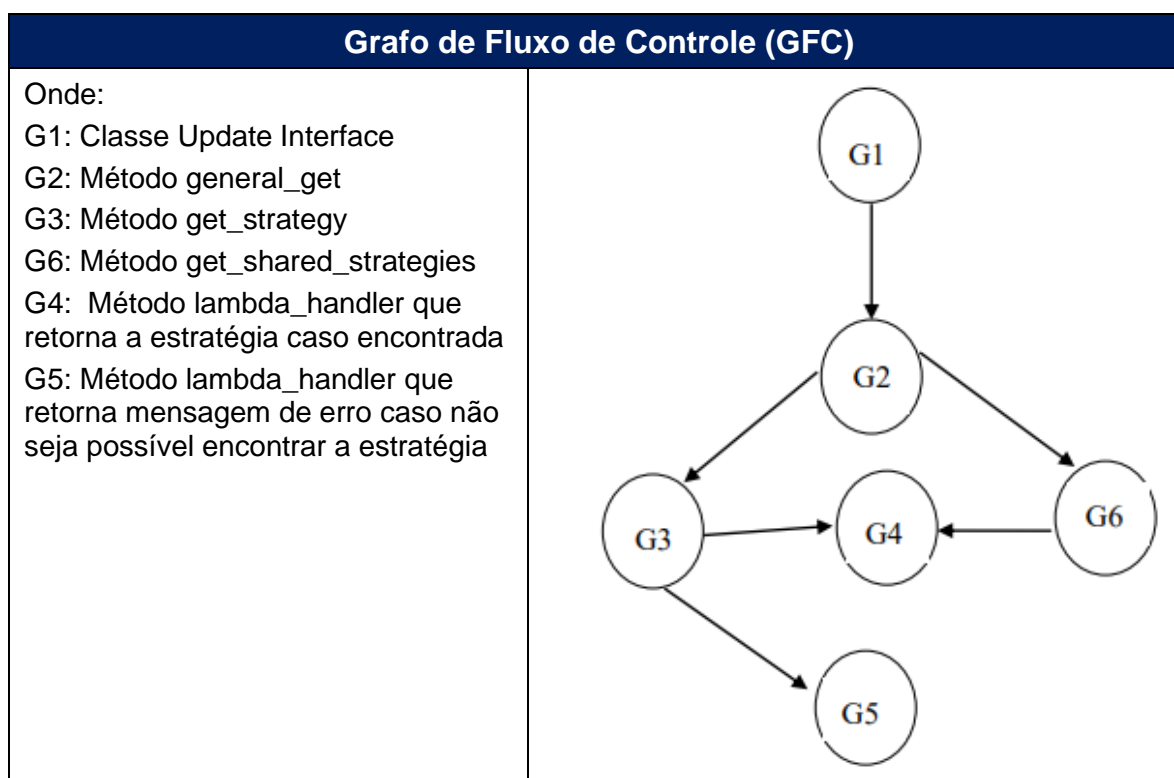
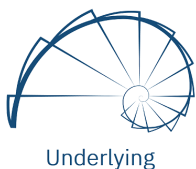


Tabela 2 - GFC para visualização da estratégia

#### 3.2 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

### 3.3 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:

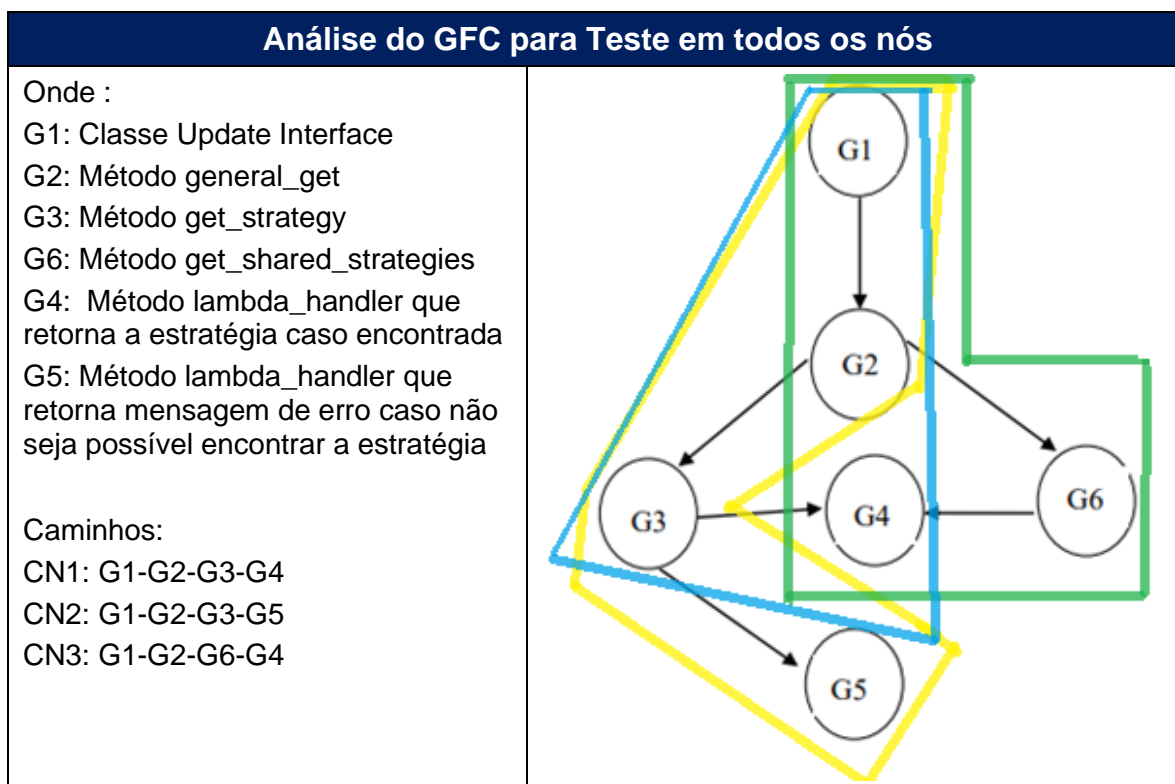


Tabela 3 - Análise do GFC para teste em todos os nós da leitura da estratégia

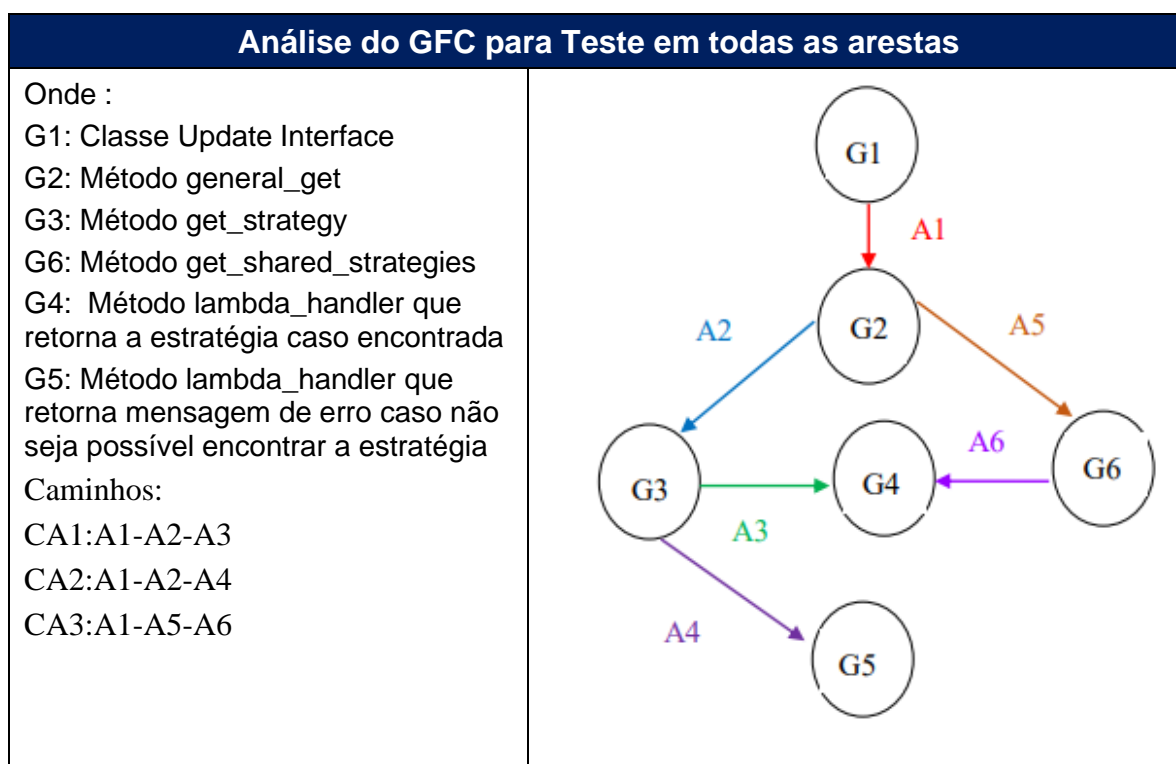


ID	CAMINHO	CASO	SAIDA
CN1	G1-G2-G3-G4	id = 18199ad0-47fb-4592-8fb2-008726efab95	assert 200 == 200 id valido (Estratégia Encontrada)
CN2	G1-G2-G3-G5	id=1234	id valido (Estratégia não Encontrada)
CN3	G1-G2-G6-G4	vazio	assert 200 == 200 (Estratégia Encontrada)

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

### 2.3. 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:



**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

CA2	A1-A2-A4	id= %4esFD12	(Estratégia Encontrada) 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**

## 2.4. Teste em todos os caminhos para visualizar estratégia

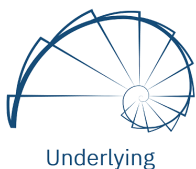
A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todas os caminhos:

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:





Underlying

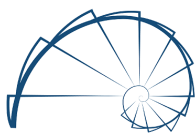


Figura 3 - Teste para Visualização das Estratégias

### 3. 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.



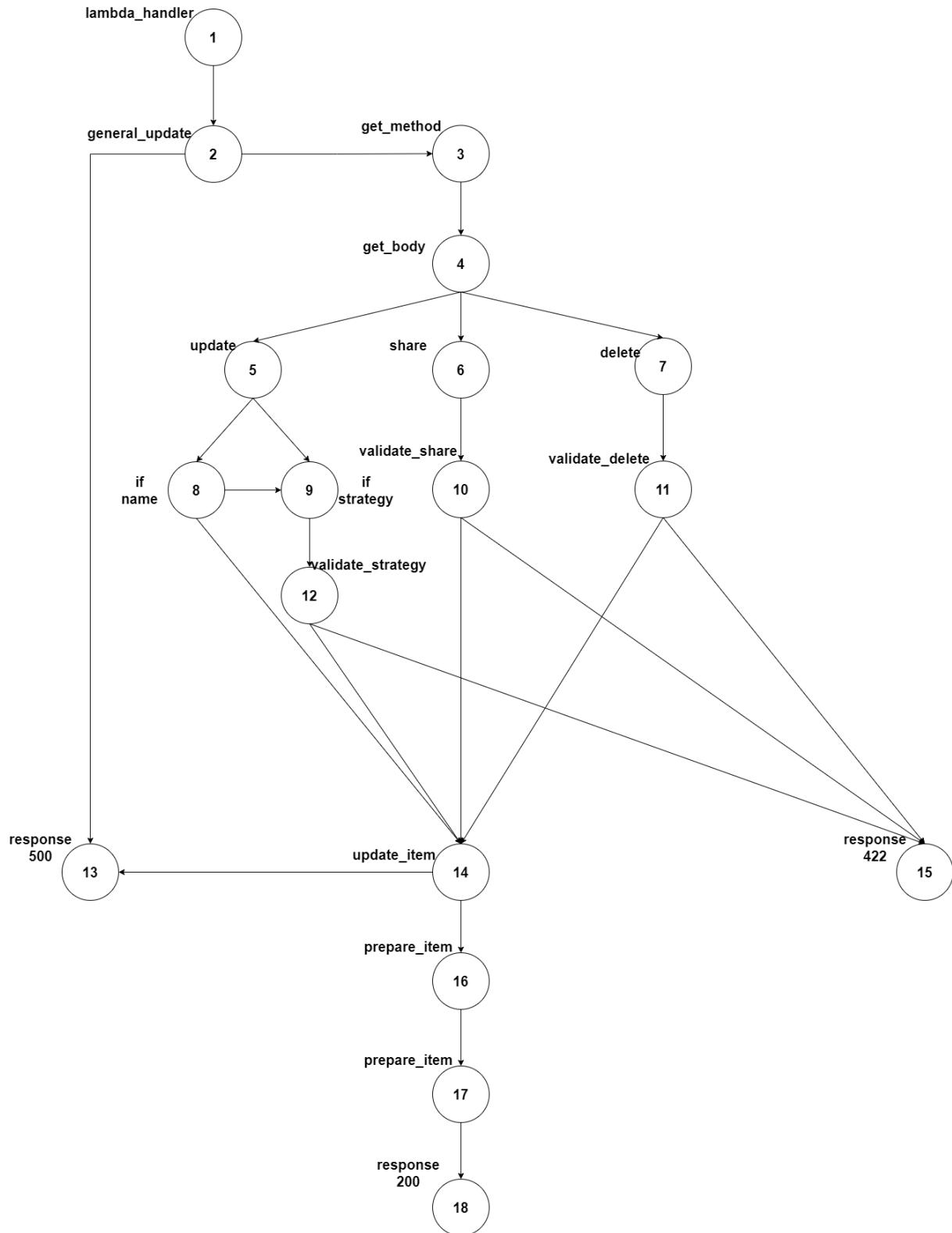
Underlying



- **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.

### 3.1. Criação do GFC para edição da estratégia

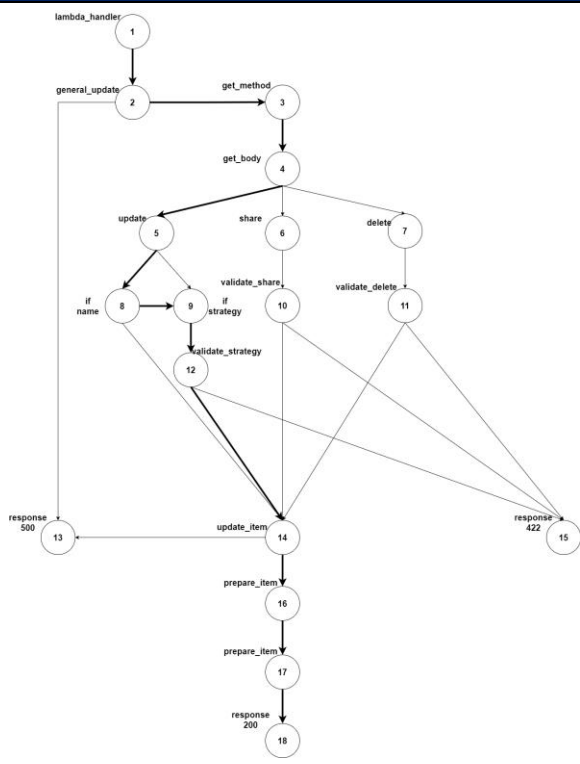
Na Tabela a seguir é apresentado o GFC para a realização da edição das estratégias criadas pelos usuários.

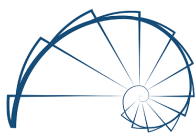


**Figura 4 - GFC para edição da estratégia**

### 3.2. 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:

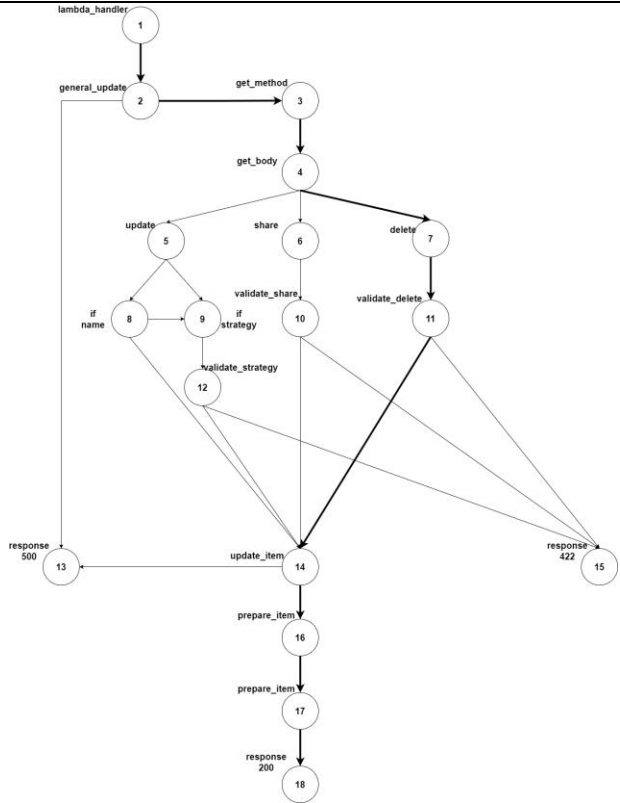
ID	CAMINHO	CASO	SAIDA
CC1	 <p><b>G1-G2-G3-G4-G5-G8-G9-G12-G14-G16-G17-G18</b></p>	<pre> rawPath = "/v1/update" body = "{   "id": "18199ad0-47fb-4592-8fb2-008726efab95",   "name": "Random Name",   "strategy": [     {       "name": "custom",       "exercise_price": 12.32,       "transaction_type": "LONG",       "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"     }   ] }"           </pre>	<p>assert 200 == 200 (Objeto atualizado)</p>

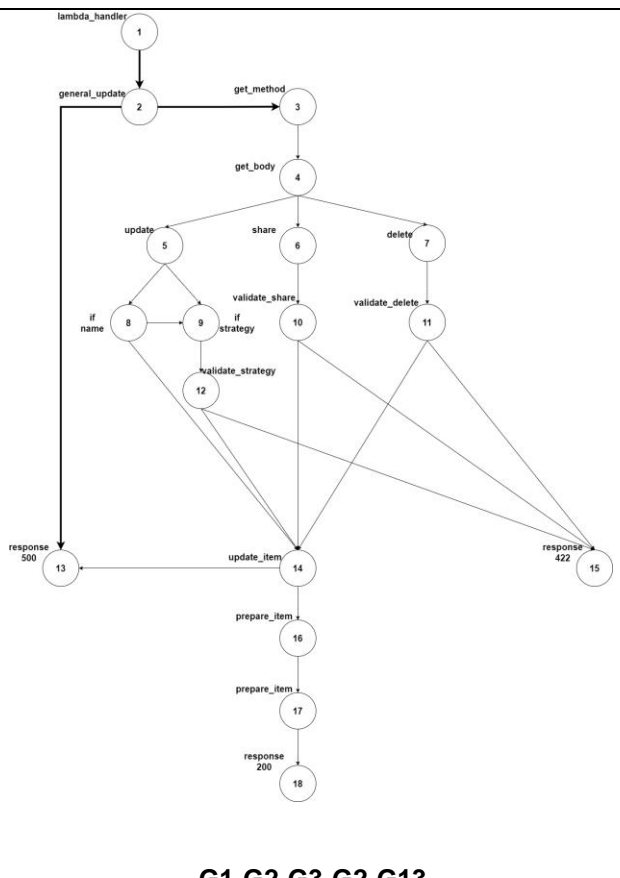


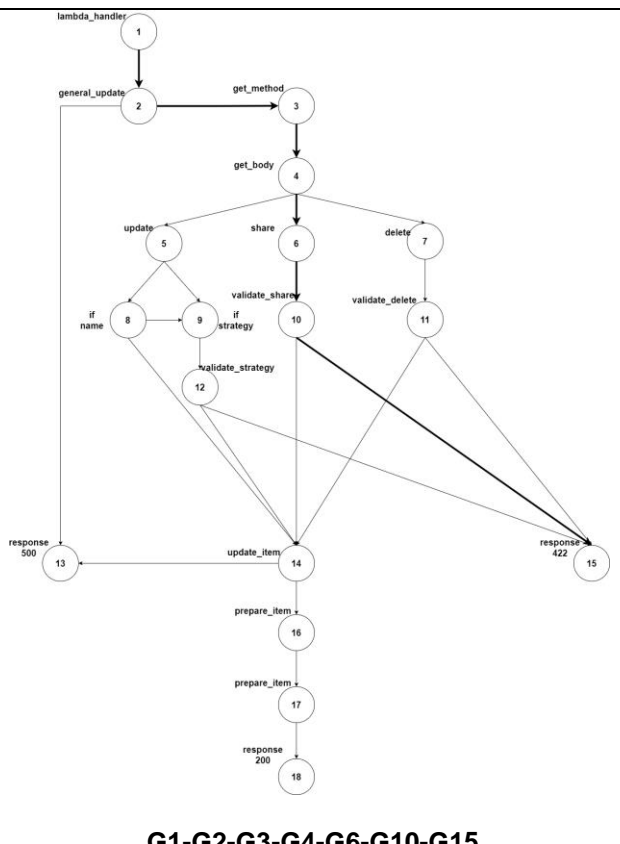
Underlying



CC2	<p><b>G1-G2-G3-G4-G6-G10-G14-G16-G17-G18</b></p>	<pre>rawPath = "/v1/share" body = "{   'id': '18199ad0-47fb-4592-8fb2-008726efab95',   'shared': True }"</pre>	<pre>assert 200 == 200 (Objeto compartilhado)</pre>
-----	--	--	---

CC3	 <p><b>G1-G2-G3-G4-G7-G11-G14-G16-G17-G18</b></p>	<pre>rawPath = "/v1/delete" body = "{   'id': '18199ad0-47fb-4592-8fb2-008726efab95',   'deleted': True }"</pre>	<pre>assert 200 == 200 (Objeto removido)</pre>
-----	---	--	--

CC4	 <p><b>G1-G2-G3-G2-G13</b></p>	rawPath = "/v1/invalid"	assert 500 == 500 (Erro – Rota não encontrada)
-----	--	-------------------------	---

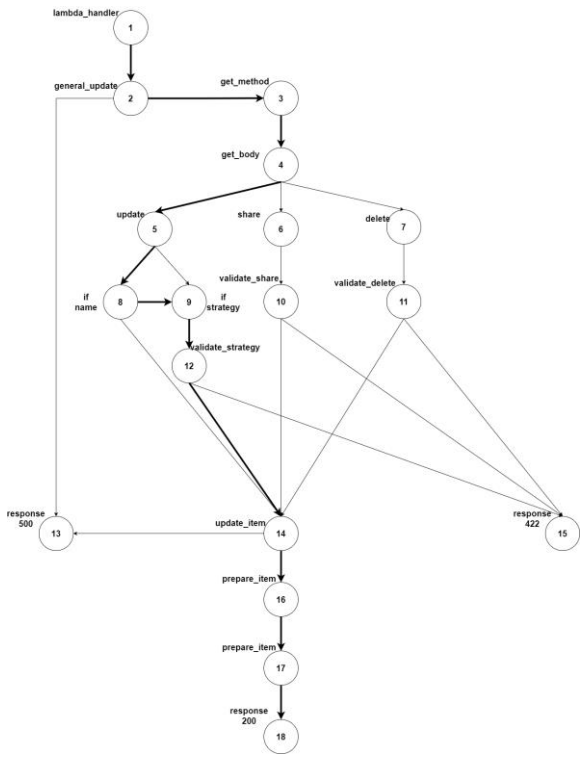
CC5	 <p><b>G1-G2-G3-G4-G6-G10-G15</b></p>	<pre>rawPath = "/v1/share" body = "{   'id': '1234' }"</pre>	<pre>assert 500 = 500 (Entrada inválida)</pre>
-----	---	--	--

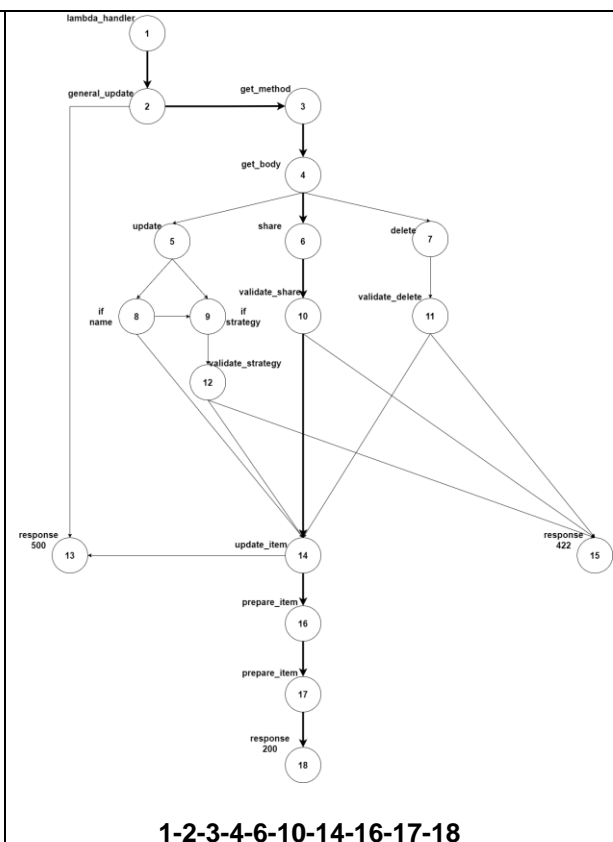
**Tabela 8 - Teste em todos os nós para atualização de estratégia**

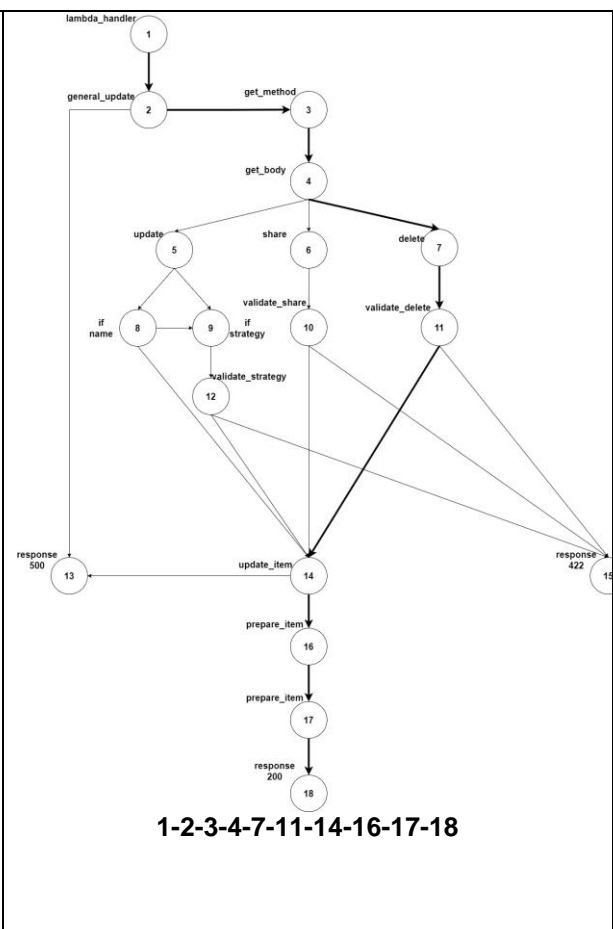
### 3.3. Teste em todas as arestas para atualização de estratégia

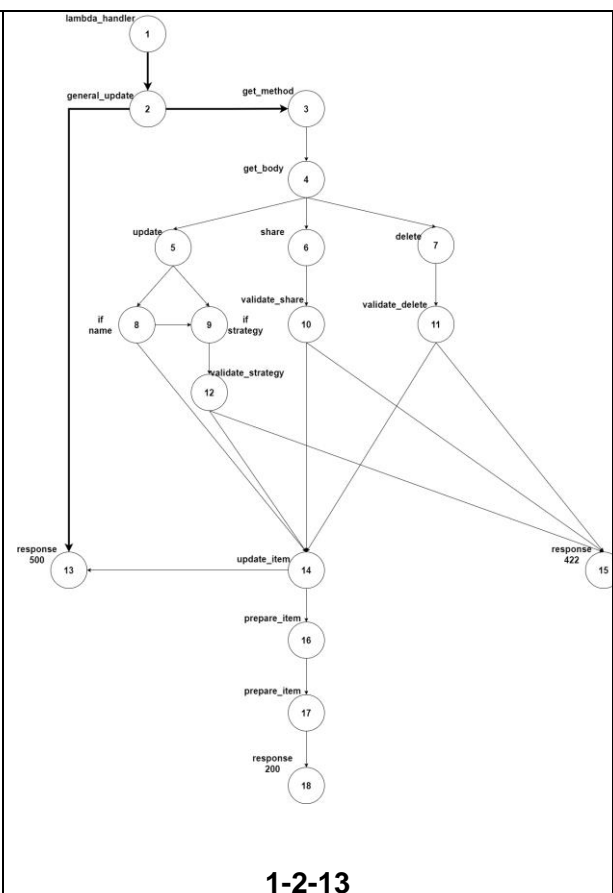
A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todos as arestas:

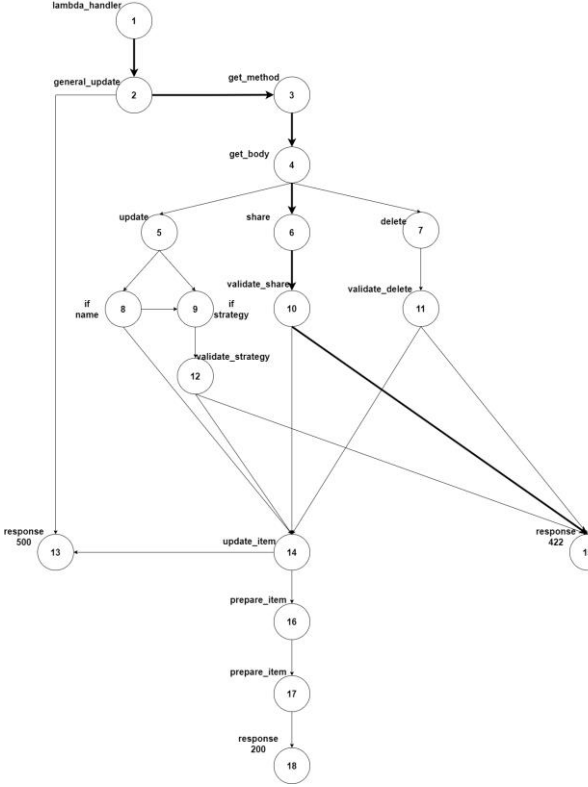
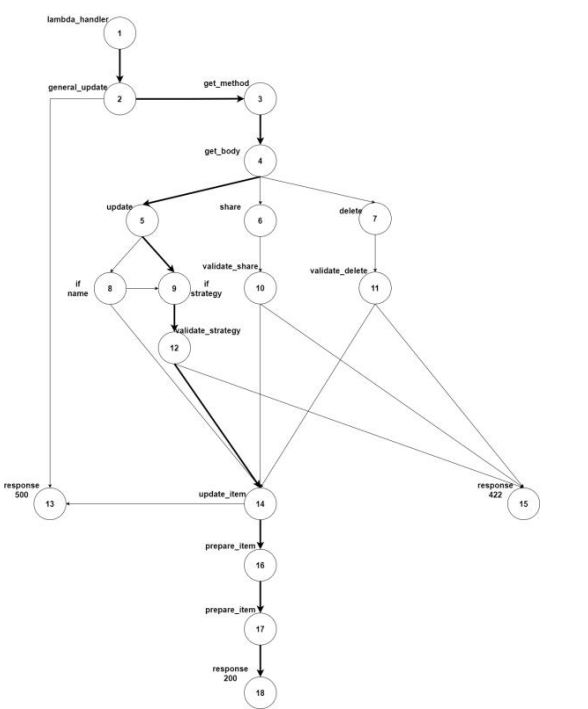


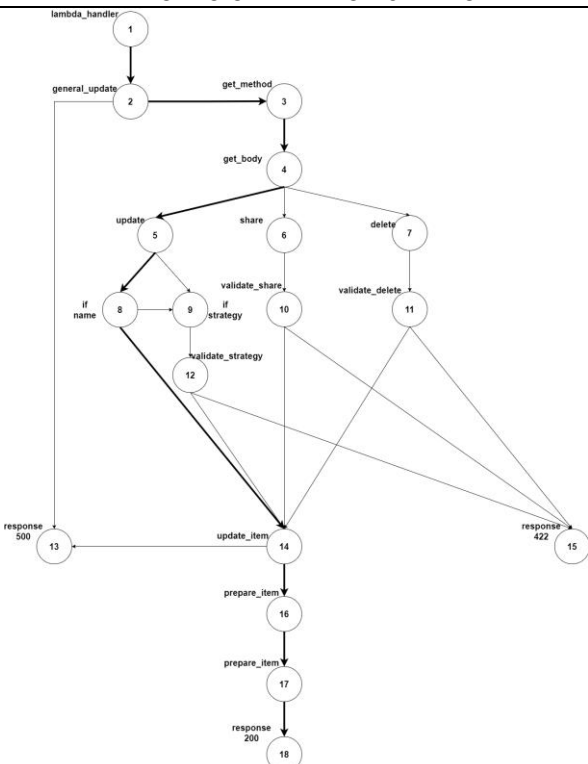
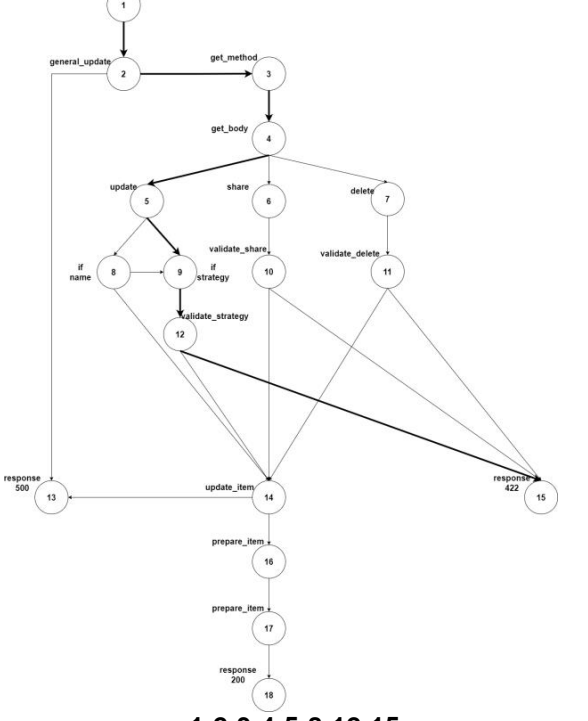
ID	CAMINHO	CASO	SAIDA
CC1	 <p><b>1-2-3-4-5-6-9-12-14-16-17-18</b></p>	<pre> rawPath = "/v1/update" body = "{   "id": "18199ad0-47fb-4592-8fb2-008726efab95",   "name": "Random Name",   "strategy": [     {       "name": "custom",       "exercise_price": 12.32,       "transaction_type": "LONG",       "close_price": 1.23,       "contracts": 1,       "type": "PUT"     },     {       "name": "custom",       "exercise_price": 12.32,       "transaction_type": "SHORT",       "close_price": 1.23,       "contracts": 1,       "type": "CALL"     }   ] }" </pre>	<pre> assert 200 == 200 (Objeto atualizado) </pre>

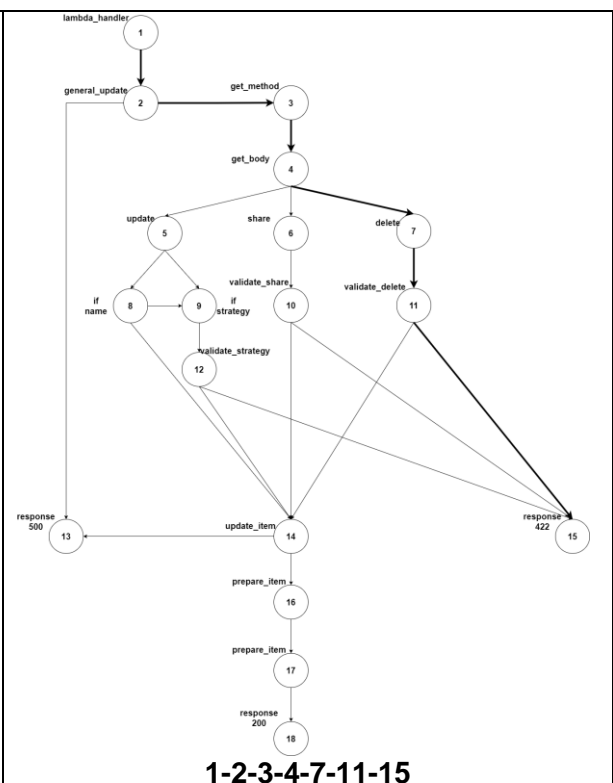
CC2	 <p><b>1-2-3-4-6-10-14-16-17-18</b></p>	<pre>rawPath = "/v1/share" body = "{   'id': '18199ad0-47fb- 4592-8fb2- 008726efab95',   'shared': True }"</pre>	<pre>assert 200 == 200 (Objeto compartilhado )</pre>
-----	---	--	--

CC3	 <pre> graph TD     1((1)) -- lambda_handler --&gt; 2((2))     2 -- general_update --&gt; 13((13))     2 --&gt; 3((3))     3 -- get_method --&gt; 4((4))     4 -- get_body --&gt; 5((5))     4 --&gt; 6((6))     4 --&gt; 7((7))     5 -- update --&gt; 8((8))     5 --&gt; 14((14))     6 -- share --&gt; 10((10))     6 --&gt; 14     7 -- delete --&gt; 11((11))     7 --&gt; 15((15))     8 -- if name --&gt; 9((9))     8 --&gt; 14     9 -- if strategy --&gt; 10     9 --&gt; 14     10 -- validate_share --&gt; 10     10 --&gt; 14     11 -- validate_delete --&gt; 11     11 --&gt; 15     12((12)) -- validate_strategy --&gt; 10     12 --&gt; 14     13 -- response 500 --&gt; 13     14 -- update_item --&gt; 14     14 --&gt; 15     14 --&gt; 16((16))     16 -- prepare_item --&gt; 17((17))     17 -- prepare_item --&gt; 18((18))     18 -- response 200 --&gt; 18     15 -- response 422 --&gt; 15 </pre> <p><b>1-2-3-4-7-11-14-16-17-18</b></p>	<pre> rawPath = "/v1/delete" body = "{   "id": "18199ad0-47fb- 4592-8fb2- 008726efab95",   "deleted": True }" </pre>	<pre> assert 200 == 200 (Objeto removido) </pre>
-----	--	--	--

CC4	 <pre> graph TD     1((1)) -- lambda_handler --&gt; 2((2))     2 -- general_update --&gt; 13((13))     2 --&gt; 3((3))     3 -- get_method --&gt; 4((4))     4 -- get_body --&gt; 5((5))     4 --&gt; 6((6))     4 --&gt; 7((7))     5 -- update --&gt; 8((8))     6 -- share --&gt; 9((9))     7 -- delete --&gt; 11((11))     8 -- if name --&gt; 9     9 -- if strategy --&gt; 10((10))     9 -- validate_strategy --&gt; 12((12))     10 -- validate_share --&gt; 14((14))     11 -- validate_delete --&gt; 15((15))     12 --&gt; 14     12 --&gt; 15     14 -- update_item --&gt; 13     14 --&gt; 15     14 --&gt; 16((16))     16 -- prepare_item --&gt; 17((17))     17 -- prepare_item --&gt; 18((18))     18 -- response 200 --&gt; 18     13 -- response 500 --&gt; 13     15 -- response 422 --&gt; 15         </pre> <p><b>1-2-13</b></p>	rawPath = "/v1/invalid"	assert 500 == 500 (Erro – Rota não encontrada)
-----	---	-------------------------	---

CC5	 <p><b>1-2-3-4-6-10-15</b></p>	rawPath = "/v1/share" body = "{ "id": "1234" }"	assert 500 = 500 (Entrada inválida)
CC6		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

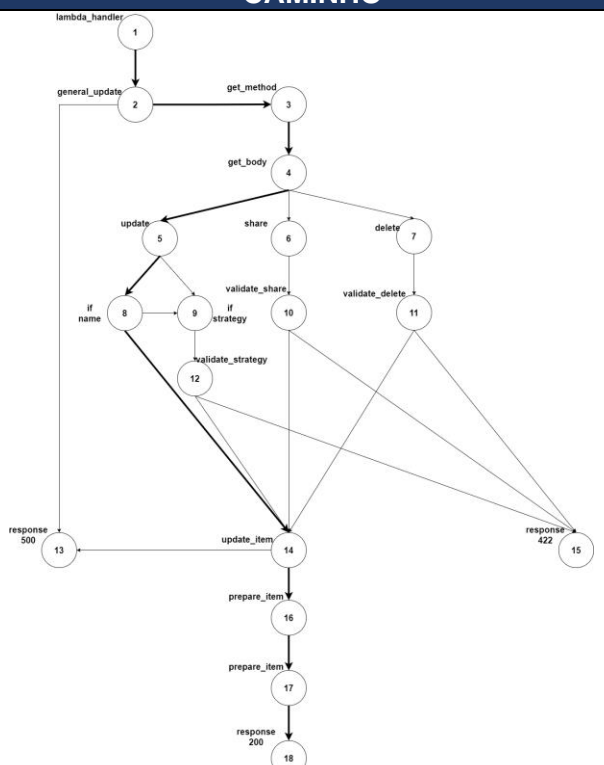
<p>CC7</p>	<p><b>1-2-3-4-5-9-12-14-15-16-17-18</b></p> 	<p>rawPath = "/v1/update" body = "{     "id": "18199ad0-47fb-4592-8fb2-008726efab95",     "name": "Random Name"}"</p>	<p>assert 200 = 200 Estratégia atualizada</p>
<p>CC8</p>	<p><b>1-2-3-4-5-8-14-15-16-17-18</b></p> 	<p>rawPath = "/v1/share" body = "{     "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"         }     ] }"</p>	<p>Assertion Error Erro na edição</p>

CC9	 <p>The graph shows a sequence of nodes starting from 'lambda_handler' (1). It branches into 'general_update' (2) and 'get_method' (3). 'get_method' leads to 'get_body' (4), which then branches into 'update' (5), 'share' (6), and 'delete' (7). 'update' leads to 'if name' (8), which leads to 'if strategy' (9). 'share' leads to 'validate_share' (10). 'delete' leads to 'validate_delete' (11). 'if strategy' (9) leads to 'validate_strategy' (12). All these paths converge at 'update_item' (14). From 'update_item' (14), there are two main paths: one leading to 'response 500' (13) and another leading to 'prepare_item' (16), which then leads to 'prepare_item' (17) and finally 'response 200' (18). There is also a direct path from 'update_item' (14) to 'response 422' (15).</p> <p><b>1-2-3-4-7-11-15</b></p>	<pre>rawPath = "/v1/delete" body = "{   'id': '18199ad0-47fb- 4592-8fb2- 008726efab95',   'deleted': 'INVALID' }"</pre>	<p>Assertion Error Erro na edição</p>
-----	--	---	---

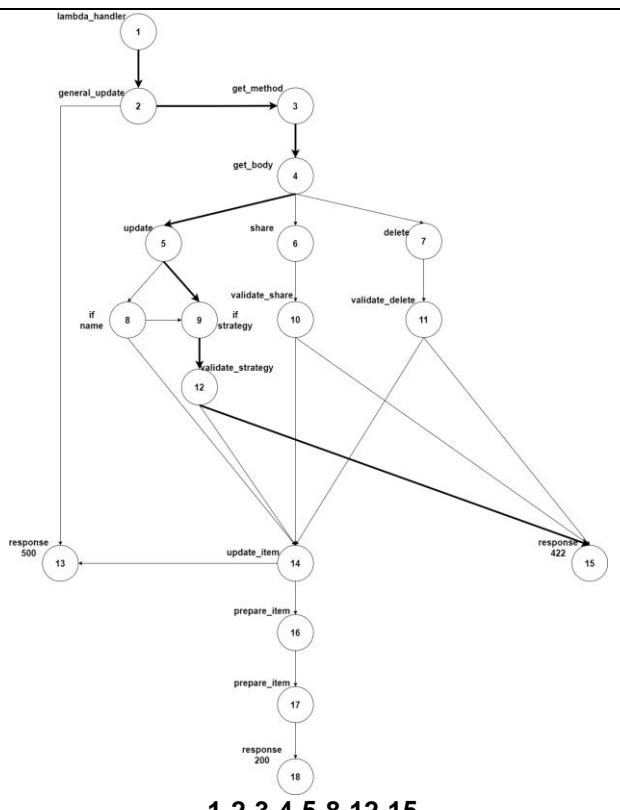
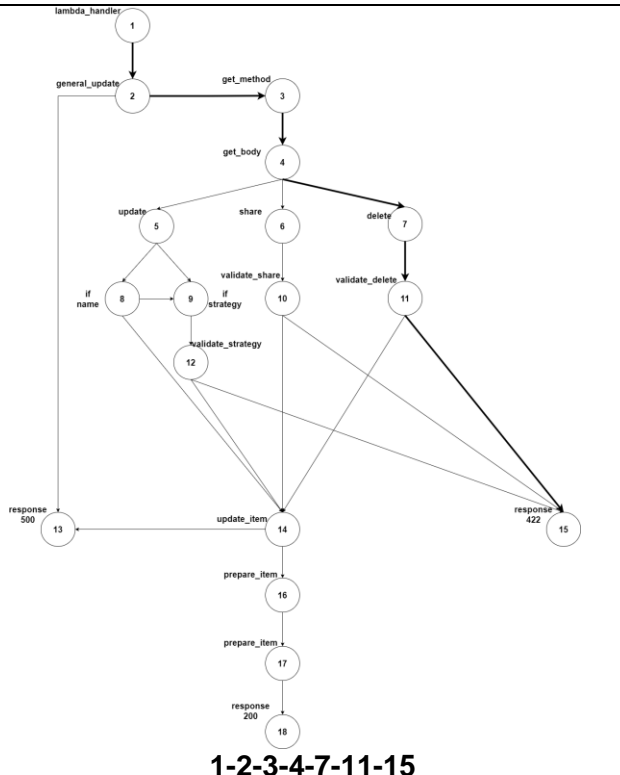
**Tabela 9 - Teste em todas as arestas da atualização da estratégia**

### 3.4. Teste em todos os caminhos para atualização de estratégia

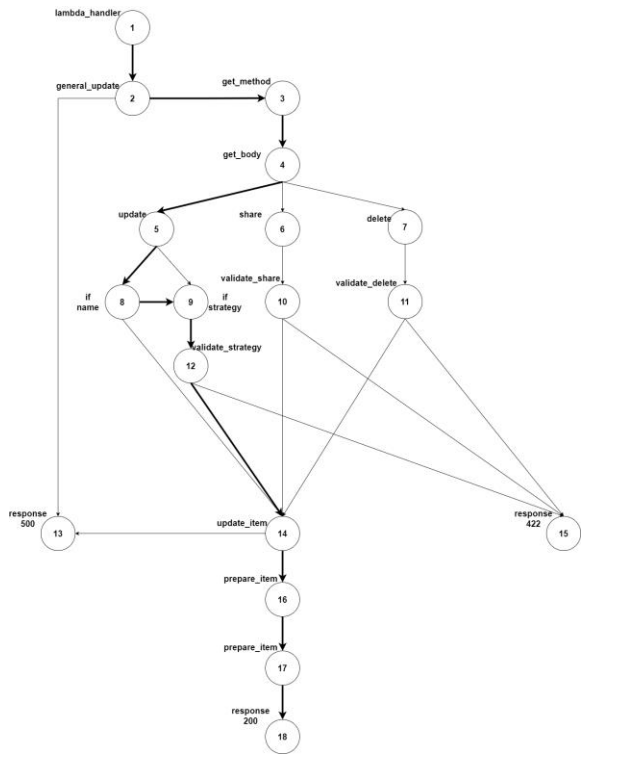
A seguir será apresentado os requisitos de teste T(P) para esse processo usando as técnicas de teste estruturais para todos os caminhos:

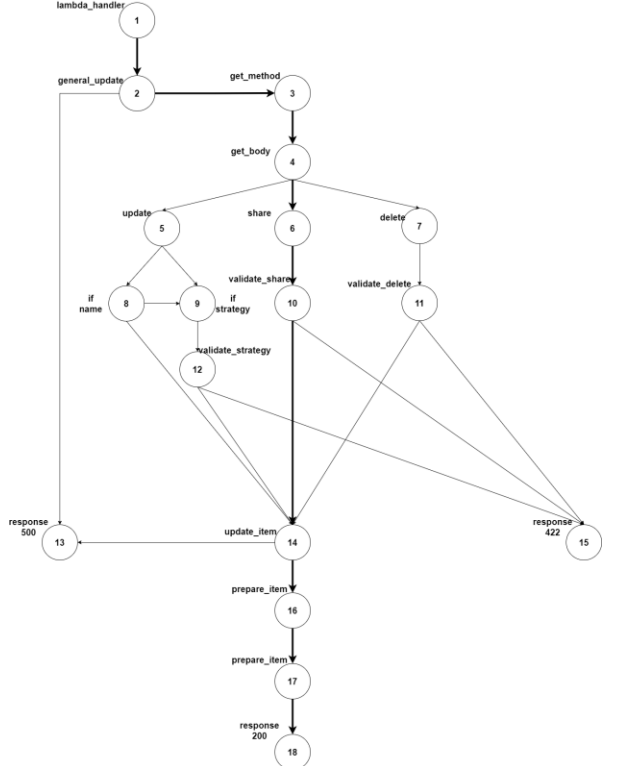
ID	CAMINHO	CASO	SAIDA
CC1	 <p><b>1-2-3-4-5-8-14-15-16-17-18</b></p>	rawPath = "/v1/update" body = "{ "id": "18199ad0- 47fb-4592-8fb2- 008726efab95", "name": "Random Name"}"	assert 200 = 200 Estratégia atualizada

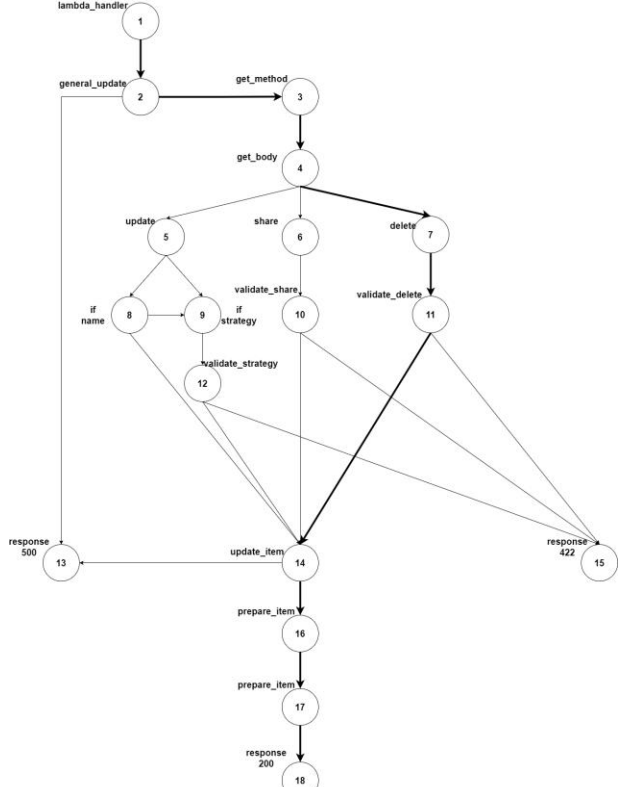


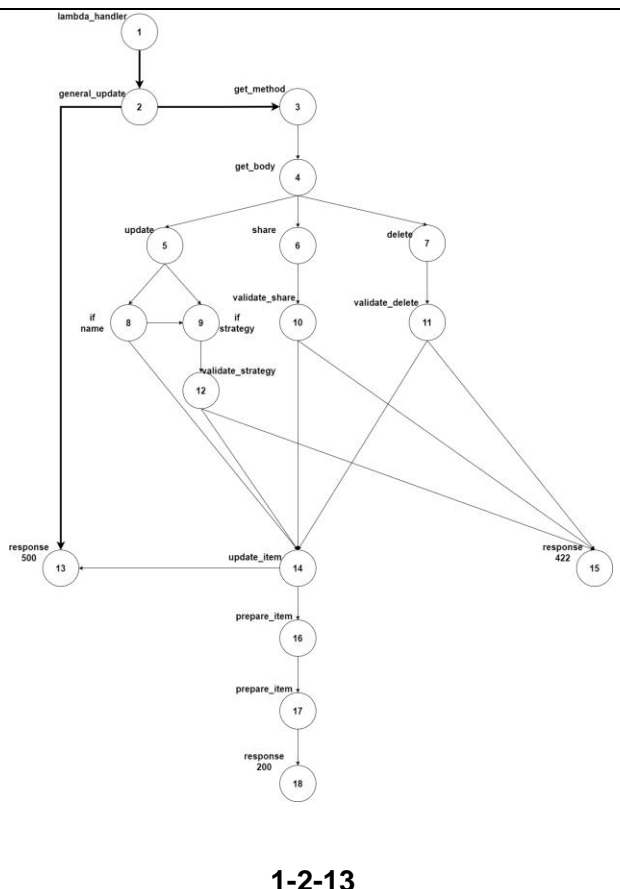
CC2	 <p><b>1-2-3-4-5-8-12-15</b></p>	<pre>rawPath = "/v1/share" body = "{   'id': '18199ad0-47fb-4592-8fb2-008726efab95',   'name': 'Random Name',   'strategy': [     {       'name': 'custom',       'exercise_price': 12.32,       'transaction_type': 'LONG',       'close_price': 1.23,       'contracts': 1,       'type': 'INVALID'     }   ] }"</pre>	<p>Assertion Error</p> <p>Erro na edição</p>
CC3	 <p><b>1-2-3-4-7-11-15</b></p>	<pre>rawPath = "/v1/delete" body = "{   'id': '18199ad0-47fb-4592-8fb2-008726efab95',   'deleted': 'INVALID' }"</pre>	<p>Assertion Error</p> <p>Erro na edição</p>

CC4	<p><b>1-2-3-4-5-9-12-14-15-16-17-18</b></p>	<pre> 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\"     }   ] }" </pre>	<p>assert 200 = 200</p> <p>Estratégia atualizada</p>
-----	---	--	--

CC5	 <p><b>1-2-3-4-5-6-9-12-14-16-17-18</b></p>	<pre> 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\"     }   ] }" </pre>	<pre> assert 200 == 200 (Objeto atualizado) </pre>
-----	---	--	--

CC6	 <pre> graph TD     1((1)) -- lambda_handler --&gt; 2((2))     2 -- general_update --&gt; 3((3))     3 -- get_method --&gt; 4((4))     4 -- get_body --&gt; 5((5))     4 -- get_body --&gt; 6((6))     4 -- get_body --&gt; 7((7))     5 -- update --&gt; 8((8))     5 -- update --&gt; 9((9))     6 -- share --&gt; 10((10))     7 -- delete --&gt; 11((11))     8 -- if name --&gt; 9     9 -- if strategy --&gt; 10     10 -- validate_strategy --&gt; 12((12))     11 -- validate_delete --&gt; 15((15))     12 -- validate_strategy --&gt; 14((14))     13((13)) -- response 500 --&gt; 14     14 -- update_item --&gt; 15     14 -- update_item --&gt; 16((16))     16 -- prepare_item --&gt; 17((17))     17 -- prepare_item --&gt; 18((18))     18 -- response 200 --&gt; 15     15 -- response 422 --&gt; 13 </pre> <p><b>1-2-3-4-6-10-14-16-17-18</b></p>	<pre> rawPath = "/v1/share" body = "{   "id": "18199ad0-47fb- 4592-8fb2- 008726efab95",   "shared": True }" </pre>	<pre> assert 200 == 200 (Objeto compartilhado ) </pre>
-----	---	--	--

CC7	 <pre> graph TD     1((1)) -- lambda_handler --&gt; 2((2))     2 -- general_update --&gt; 13((13))     2 --&gt; 3((3))     3 -- get_method --&gt; 4((4))     4 -- get_body --&gt; 5((5))     4 --&gt; 6((6))     4 --&gt; 7((7))     5 -- update --&gt; 8((8))     5 --&gt; 14((14))     6 -- share --&gt; 10((10))     6 --&gt; 14     7 -- delete --&gt; 11((11))     7 --&gt; 15((15))     8 -- if name --&gt; 9((9))     9 -- if strategy --&gt; 10     9 --&gt; 12((12))     10 -- validate_share --&gt; 14     11 -- validate_delete --&gt; 15     12 -- validate_strategy --&gt; 14     13 -- response 500 --&gt; 13     14 -- update_item --&gt; 13     14 --&gt; 15     14 --&gt; 16((16))     16 -- prepare_item --&gt; 17((17))     17 -- prepare_item --&gt; 18((18))     18 -- response 200 --&gt; 18     15 -- response 422 --&gt; 15 </pre> <p><b>1-2-3-4-7-11-14-16-17-18</b></p>	<pre> rawPath = "/v1/delete" body = "{   "id": "18199ad0-47fb- 4592-8fb2- 008726efab95",   "deleted": True }" </pre>	<pre> assert 200 == 200 (Objeto removido) </pre>
-----	---	--	--

CC8	 <p>The graph shows a sequence of operations starting from a lambda handler. It branches into a 'general_update' path leading to a 500 response, and a 'get_method' path leading to 'get_body'. From 'get_body', it branches into 'update', 'share', and 'delete'. The 'update' path involves conditional checks and validation before reaching an 'update_item' node. The 'share' and 'delete' paths also lead to validation and then to 'update_item'. The 'update_item' node leads to 'prepare_item', which eventually results in a 200 response. A specific path is highlighted from node 1 to 2 to 13.</p> <p><b>1-2-13</b></p>	rawPath = "/v1/invalid"	assert 500 == 500 (Erro – Rota não encontrada)
-----	--	-------------------------	---

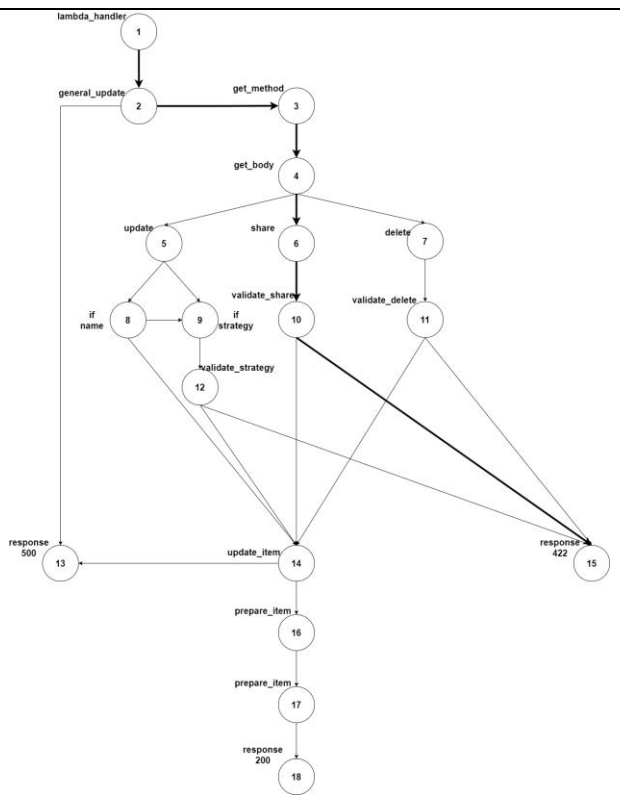
CC9	 <p>1-2-3-4-6-10-15</p>	<pre>rawPath = "/v1/share" body = "{   'id': '1234' }"</pre>	<p>assert 500 = 500 (Entrada inválida)</p>
-----	---	--	--

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

## 4. 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.

### 4.1. 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 ".".

#### 4.1.1. Função de validação do email

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

```

1  function validaEmail() {
2      if(this.state.email != null && this.state.email != ""){
3          //Se não houver @ na string, retorna falso
4          if(this.state.email.indexOf('@') == -1){
5              this.setState({
6                  emailError: "Email inválido! É preciso haver um @"
7              });
8              return false;
9          } else { //Se não houver . na string, retorna falso
10             if(this.state.email.indexOf('.') == -1){
11                 this.setState({
12                     emailError: "Email inválido! É preciso haver um ."
13                 });
14                 return false;
15             } else {
16                 var dotIndex = this.state.email.indexOf('.');
17                 if(typeof(this.state.email[dotIndex + 1]) != 'string') {
18                     this.setState({
19                         emailError: "Email inválido! É preciso haver um domínio de email completo"
20                     });
21                     return false;
22                 } else {
23                     return true;
24                 }
25             }
26         }
27     } else return false;
28 }

```

Figura 5 - Função de validação do email

#### 4.1.2. GFC para validação do Email

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

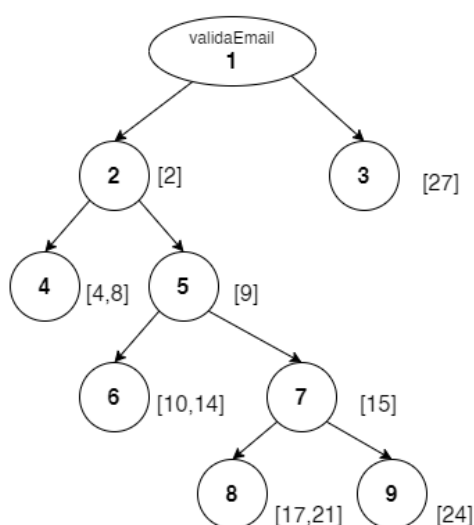


Figura 6 - GFC para validação do Email



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

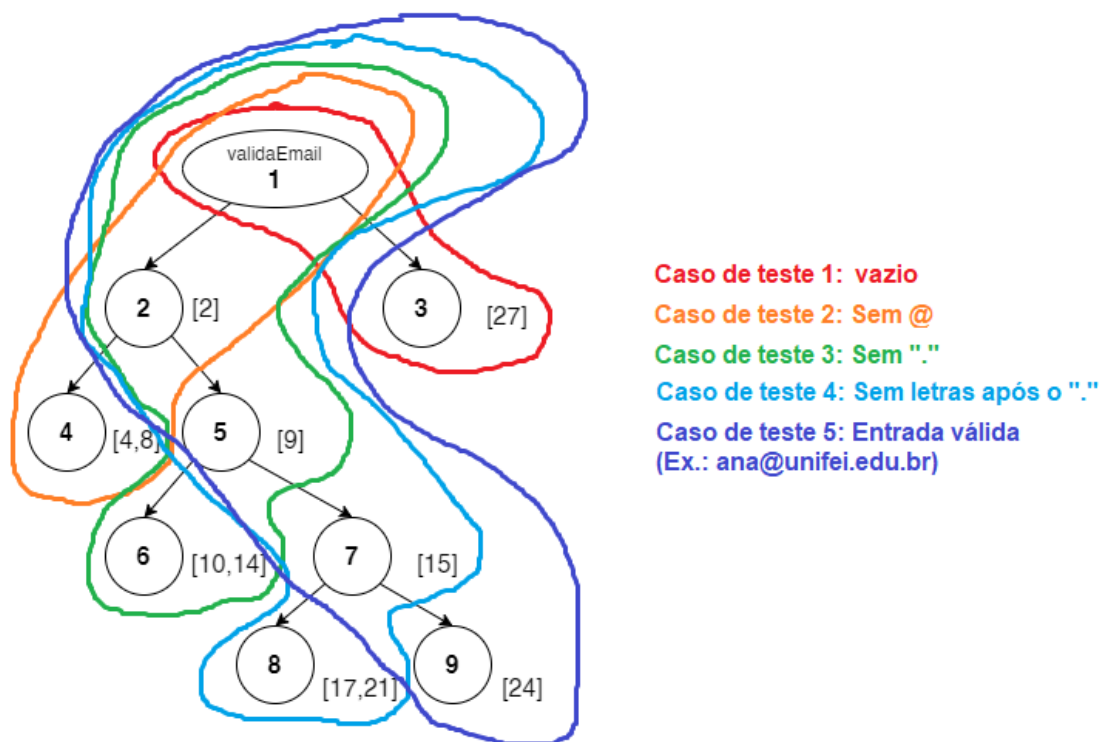


Figura 7 - Análise de todos os nós para validar email

#### 4.1.4. 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

## 4.2. 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;
- 

### 4.2.1. Função de validação do nome do usuário

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

```
1  function validaUser() {
2      if (this.state.user !== "") {
3          if (this.state.user.length >= 3) {
4              var count = this.state.user.split(" ").length;
5              if (count > 1) {
6                  return true;
7              } else {
8                  this.setState({
9                      userError: "Usuário deve ter mais de 2 palavras"
10                 })
11                 return false;
12             }
13         } else {
14             this.setState({
15                 userError: "Usuário deve ter mais de 3 caracteres"
16             })
17             return false;
18         }
19     } else {
20         this.setState({
21             userError: "Campo de usuário é obrigatório"
22         })
23         return false;
24     }
25 }
```

Figura 8 - Função de validação do nome do usuário

### 4.2.2. 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:

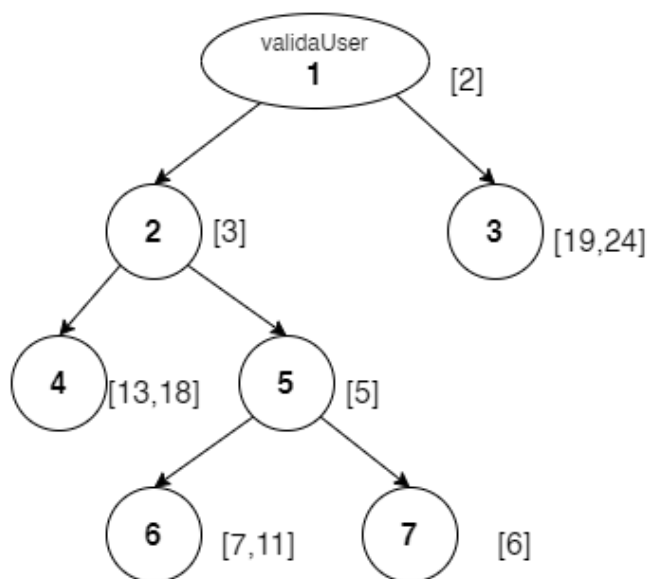


Figura 9 - FGC para validação do nome do usuário

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

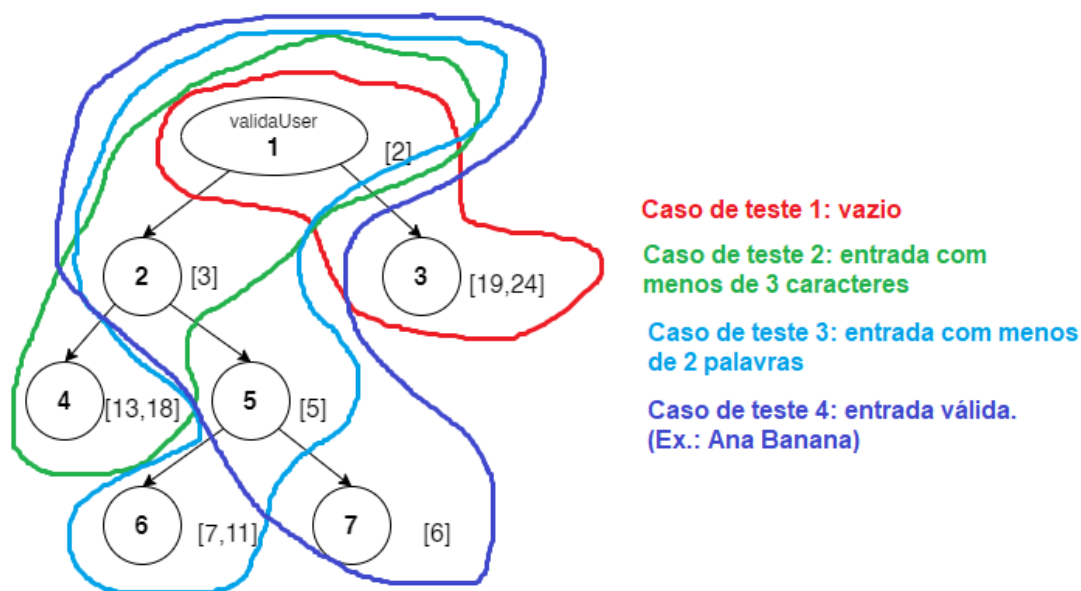
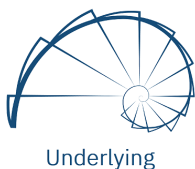


Figura 10 - Análise de todos os nós para nome do usuário

#### 4.2.4. 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**

## 5. TESTE DE CALCULO DE PAYOFF

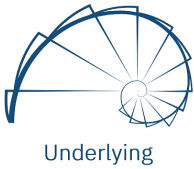
O cálculo de payoff para estratégias é diferenciado a partir de campos `type` e `transaction_type`. O Cálculo será então realizado em cima do `exercise_price` e `close_price`, considerando que o retorno é 200 para o teste passar e 500 em erro interno.

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;

### 5.1. Função de cálculo de Payoff

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



Underlying



```
def single_payoff(option, x):
    y = []
    scenarios = len(x)
    if option["type"].upper() == 'CALL':
        for i in range(scenarios):
            y.append(max((x[i] - option["exercise_price"]
                          - option["close_price"]), -option["close_price"]))
    else:
        for i in range(scenarios):
            y.append(max(option["exercise_price"] - x[i]
                          - option["close_price"], -option["close_price"]))

    y = np.array(y)
    y = -y if option["transaction_type"] == "SHORT" else y
    return {**option, "x": x, "y": y * option["contracts"]}
```

Figura 11 - Função de cálculo de Payoff

## 5.2. GFC para Calculo de Payoff

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

```
def lambda_handler(event, context):
    try:
        event = json.loads(event["body"])
        mean_price = np.mean([o["exercise_price"] for o in event["strategy"]])
        x = get_x(mean_price)

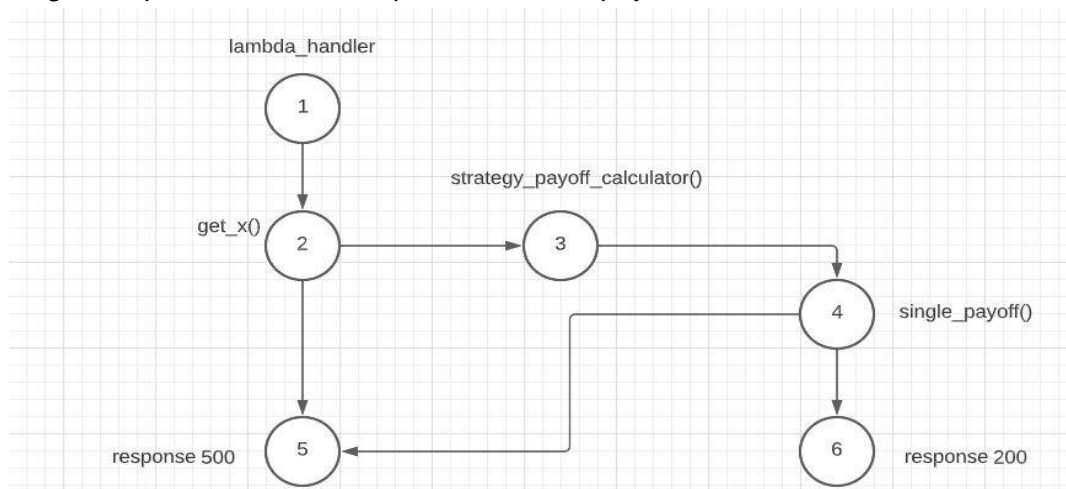
        result = strategy_payoff_calculator(event["strategy"], x)

        return {
            "statusCode": 200,
            "body": json.dumps(result, cls=NumpyArrayEncoder)
        }

    except Exception as e:
        print(e)
        return {
            "statusCode": 500,
            "body": f"Internal Server Error"
        }
```

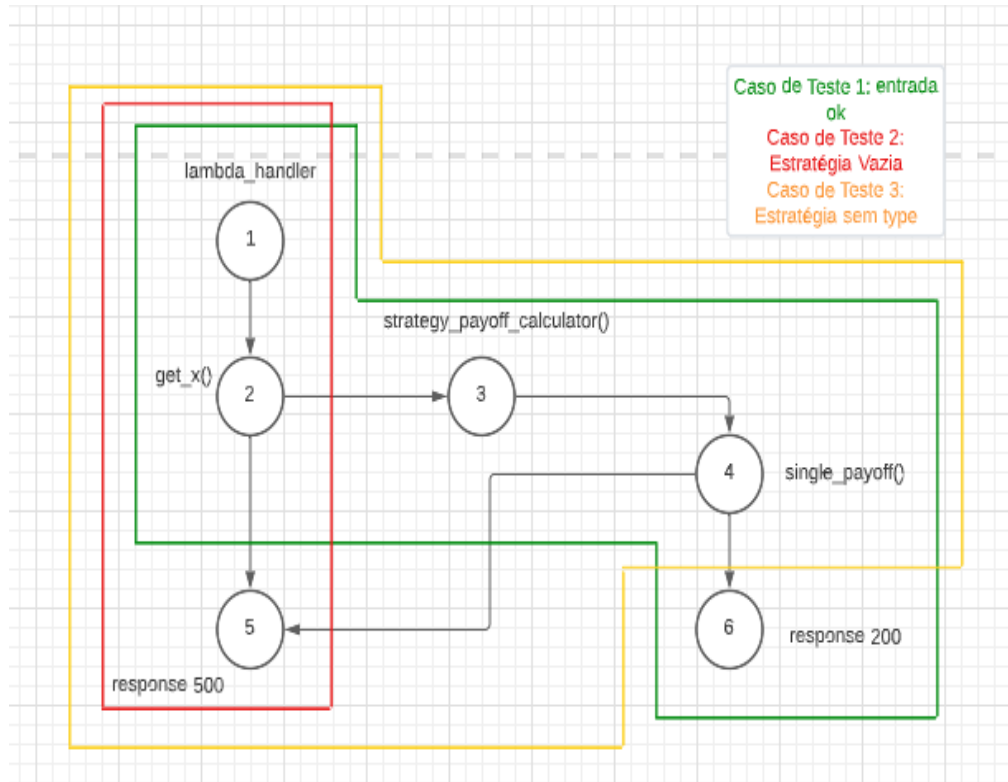
Figura 12 - Função lambda\_handler

A seguir é apresentado o GFC para cálculo de payoff:



**Figura 13 - GFC para Cálculo de Payoff**

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

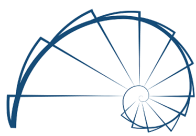


**Figura 14 - Análise de todos os nós para validar o cálculo de Payoff**

#### 5.4. 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	<pre>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,</pre>	<pre>{   "statusCode": 200,   "body": json.dumps(result, cls=NumpyArrayEncoder) }</pre>



Underlying



		<pre>         "type": "PUT"       },       {         "name": "custom", "exercise_price": 12.32, "transaction_type": "SHORT", "close_price": 1.23, "contracts": 1, "type": "CALL"       }     ]   }) }</pre>	
CT2	1-2,2-5	base_case: {}	<pre> {   "statusCode": 500,   "body": f"Internal Server Error" }</pre>
CT3	1-2, 2-3,3-4,4-5	<pre> 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,</pre>	<pre> {   "statusCode": 500,   "body": f"Internal Server Error" }</pre>

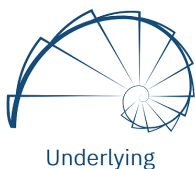


		<pre> "transaction_type": "LONG",     "close_price": 1.23,     "contracts": 1,     "type": "CALL"     },     {         "name": "custom", "exercise_price": 12.32,  "transaction_type": "SHORT",     "close_price": 1.23,     "contracts": 1,     "type": "PUT"     },     {         "name": "custom", "exercise_price": 12.32,  "transaction_type": "SHORT",     "close_price": 1.23,     "contracts": 1,     "type": "CALL"     }     ]     })     } </pre>	
--	--	--	--

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

## 6. 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.



## 6.1. 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 ( . )

### 6.1.1. Código para a validação do campo de Email

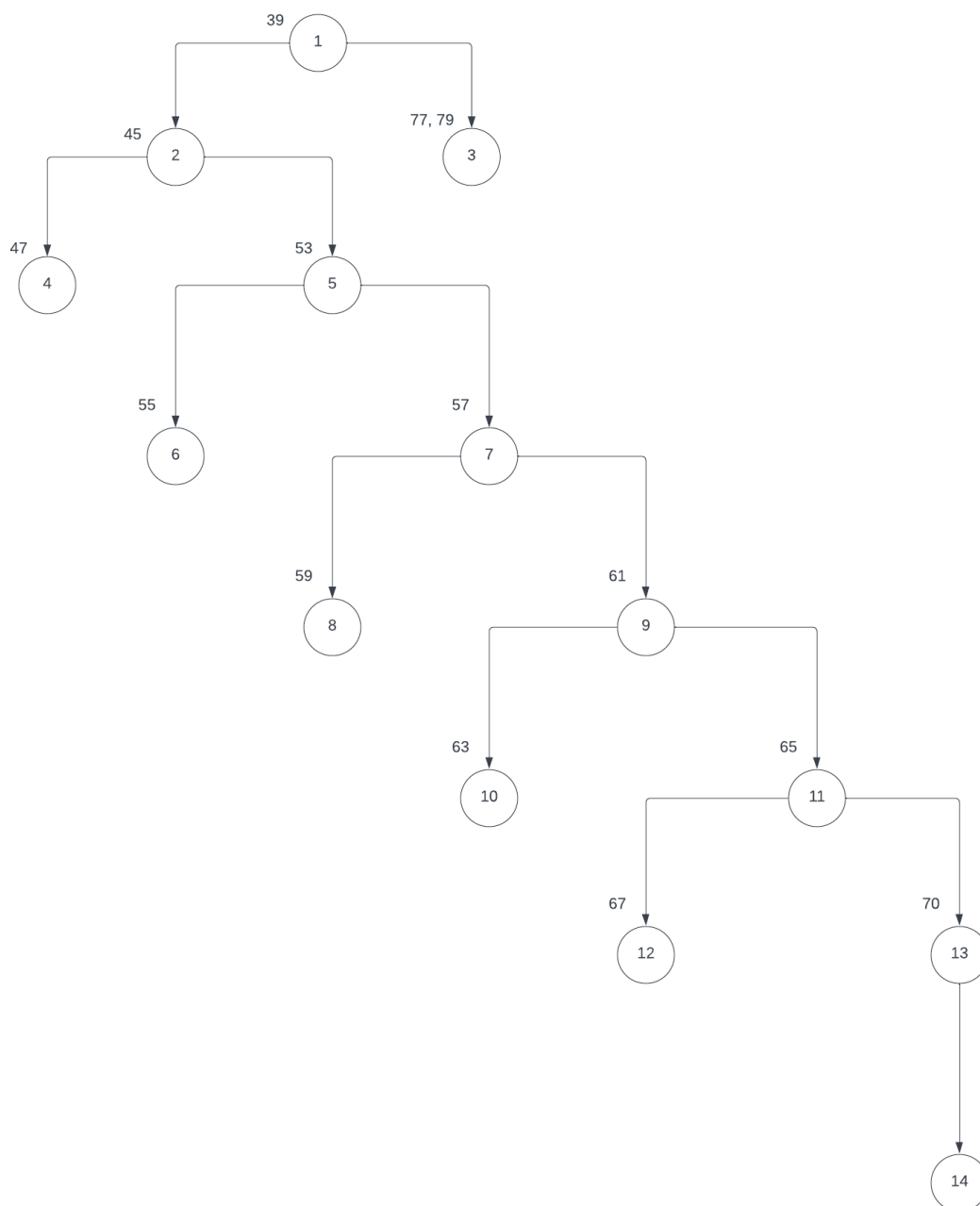
O trecho de código abaixo é responsável por garantir a validade das informações inseridas no campo Email, seguindo os parametros estabelecidos no tópico 5.1.2

```
validaEmail(email){  
  
    //Valida se o email não está vazio  
    if(email != '' && email != null){  
  
        //Verifica a existencia do caracter @  
        if(email.indexOf('@') == -1){  
            console.log('Email não possui o caractere @')  
            return false  
        }  
        else{  
            const emailDividido = email.split("@")  
  
            //Verificação do tamanho do email  
            if(emailDividido[0].length < 3){  
                console.log("O email não possui a quantidade mínima de caracteres antes do caractere @")  
                return false  
            }  
            else if(emailDividido[0].length > 30){  
                console.log("o email excede a quantidade máxima de caracteres antes do caractere @")  
                return false  
            }  
            if(emailDividido[1].length < 3){  
                console.log("O email não possui a quantidade mínima de caracteres depois do caractere @")  
                return false  
            }  
            else if(emailDividido[1].length > 30){  
                console.log("o email excede a quantidade máxima de caracteres depois do caractere @")  
                return false  
            }  
            else{  
                console.log("O email atende os requisitos")  
                return true  
            }  
        }  
    }  
}
```

Figura 15 - Código para a validação do campo de Email

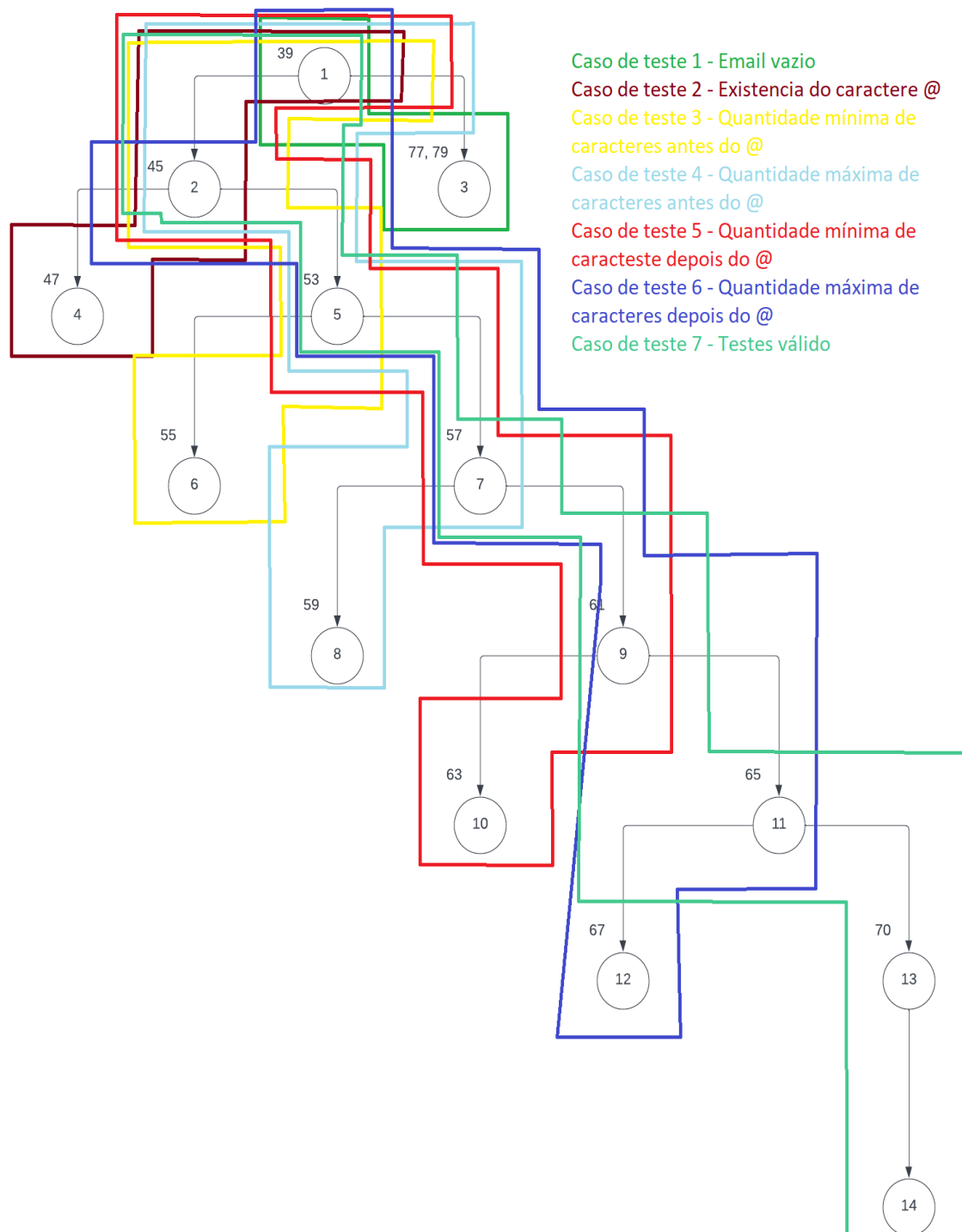
### 6.1.2. FGC para validar campo de email

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**

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



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

#### 6.1.4. 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 caractere '@' 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 caractere '@'
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

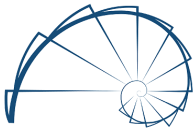
Tabela 14 - Análise de todas as arestas para validar campo de email

#### 6.2. 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

##### 6.2.1. 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



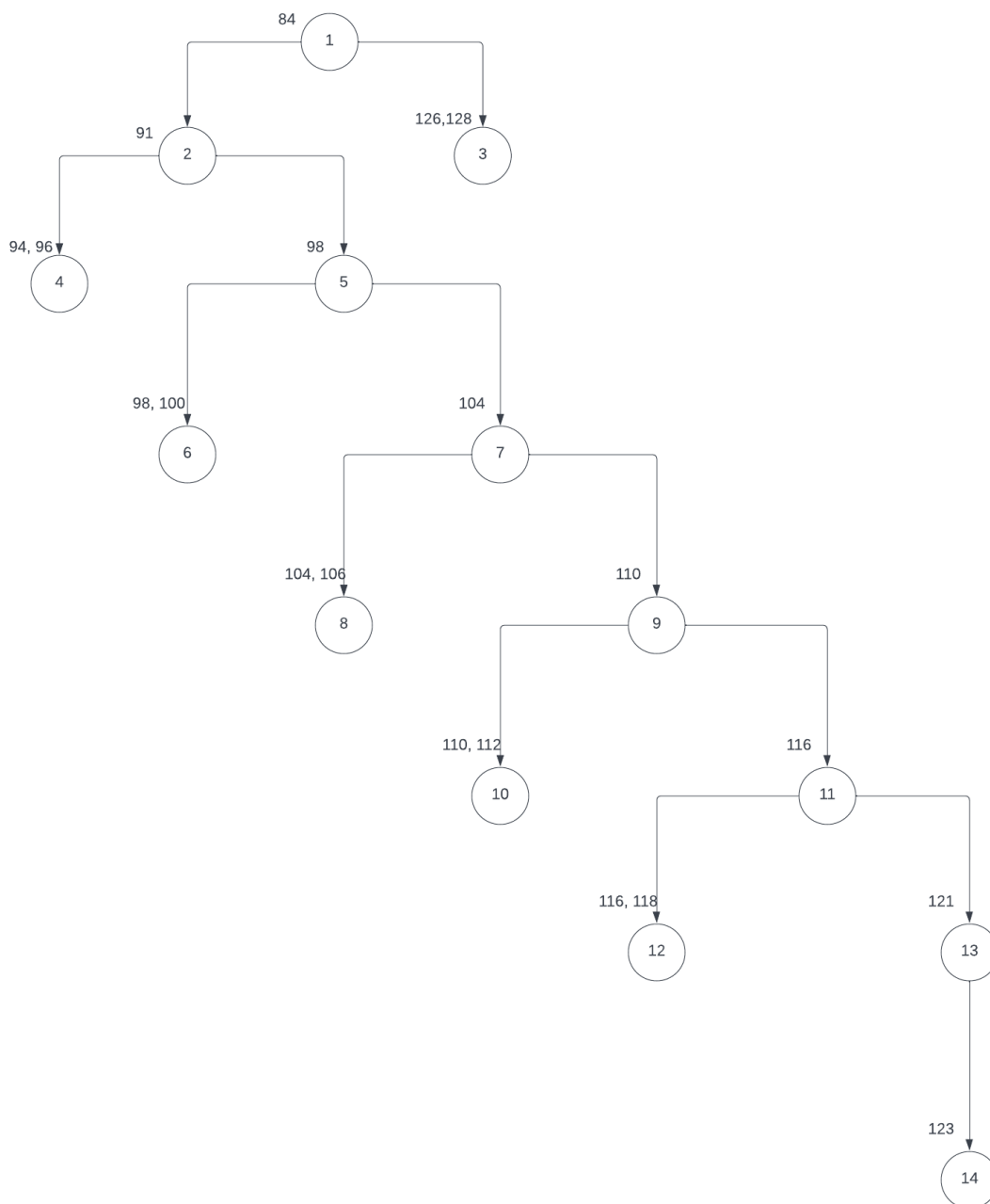
Underlying

```
validaSenha(senha){  
  
    //Regex para validar a existencia de caracteres especiais  
    const regexCaracteresEspeciais = /\W|_|/  
    const regexNumero = /[0-9]/  
    const regexLetras = /[a-z,A-Z]/  
  
    if(senha != '' && senha != null){  
  
        //Verificação do tamanho da senha  
        if(senha.length < 8){  
            console.log("A senha não possui a quantidade mínima de caracteres")  
            return false  
        }  
        else if(senha.length > 20){  
            console.log("A senha excede a quantidade máxima de caracteres")  
            return false  
        }  
  
        //Verifica a existencia de um caractere especial  
        else if(!regexCaracteresEspeciais.test(senha)){  
            console.log("A senha não possui a existencia de caracteres especiais")  
            return false  
        }  
  
        //Verifica a existencia de um número  
        else if(!regexNumero.test(senha)){  
            console.log("A senha não possui a quantidade mínima de números")  
            return false  
        }  
  
        //Verifica a existencia de uma letra  
        else if(!regexLetras.test(senha)){  
            console.log("A senha não possui a quantidade mínima de letras")  
            return false  
        }  
  
        else{  
            console.log("A senha atende os requisitos")  
            return true  
        }  
    }  
    else{  
        console.log("O campo de senha está vazio")  
        return false  
    }  
}
```

Figura 18 - Código para a validação do campo de Senha

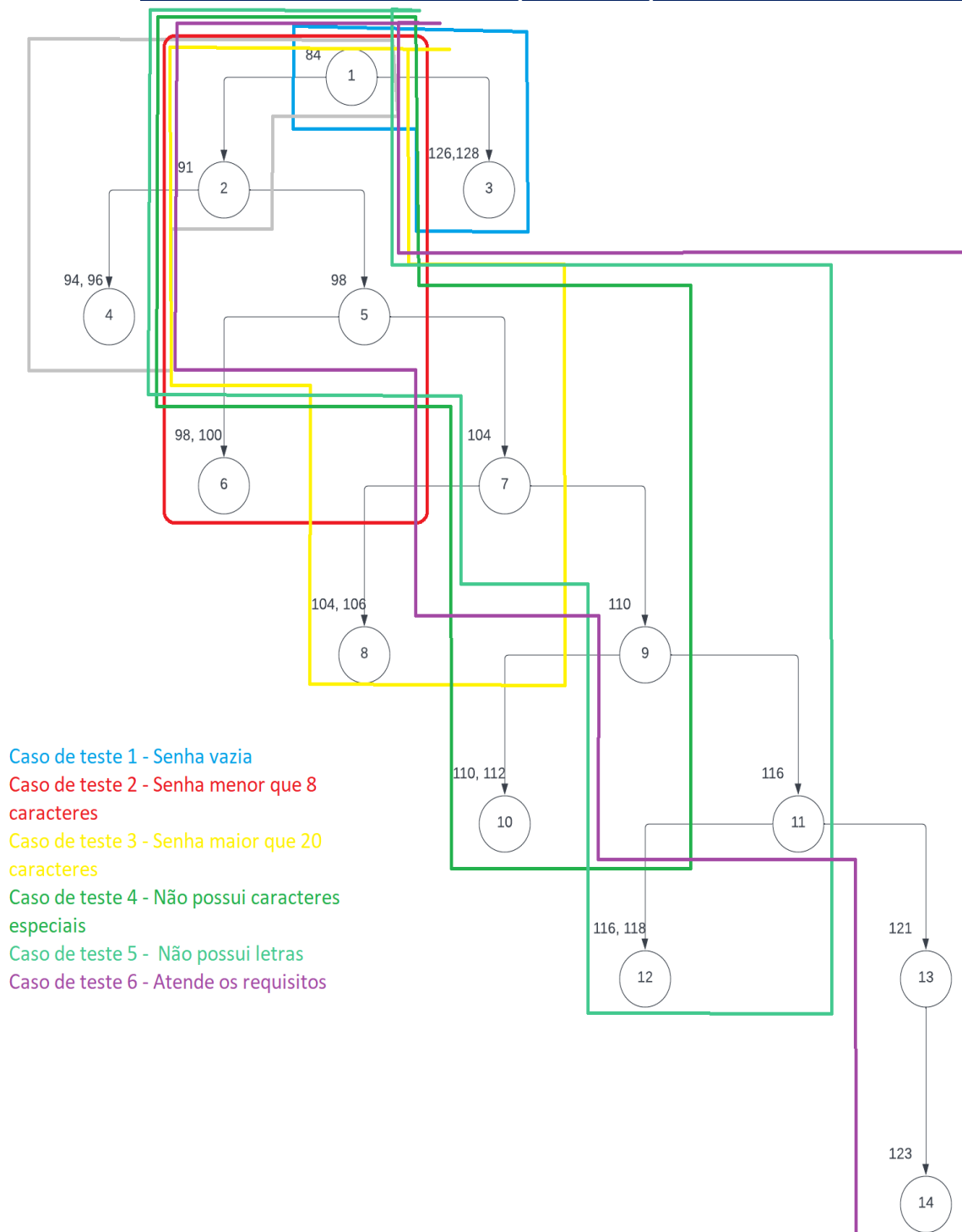
### 6.2.2. GFC do campo de senha

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**

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



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



#### 6.2.4. 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

## 7. 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

### 7.1. 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

## 7.2. 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:

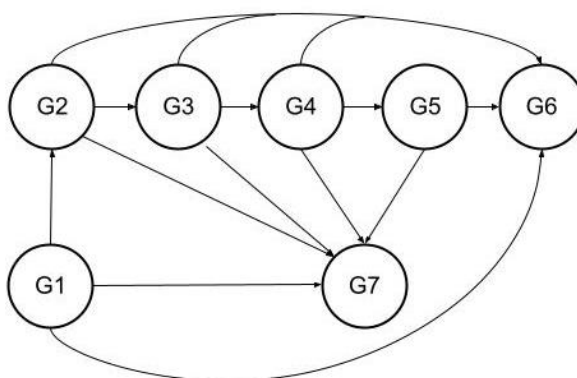


Figura 21 - GFC para busca de opções

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

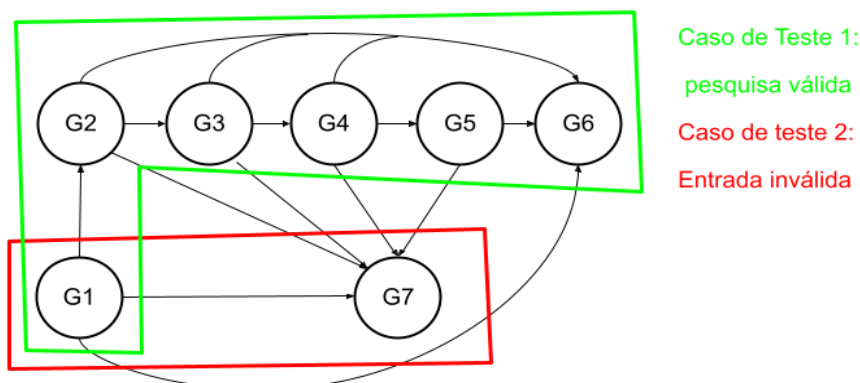
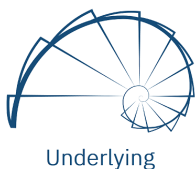


Figura 22 - Análise de todos os nós para busca de opções



#### 7.4. 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

## 8. 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

### 8.1. Função de busca de opção backend

Segue a função de busca de opção

```
async def get_option(option):
    try:
        return pd.read_parquet(f"s3://{SERIES_BUCKET}/name={option['name']}/{option['id']}.snappy.parquet")
    except Exception as e:
        raise e
```

Figura 23 - Busca de opção (backend)

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

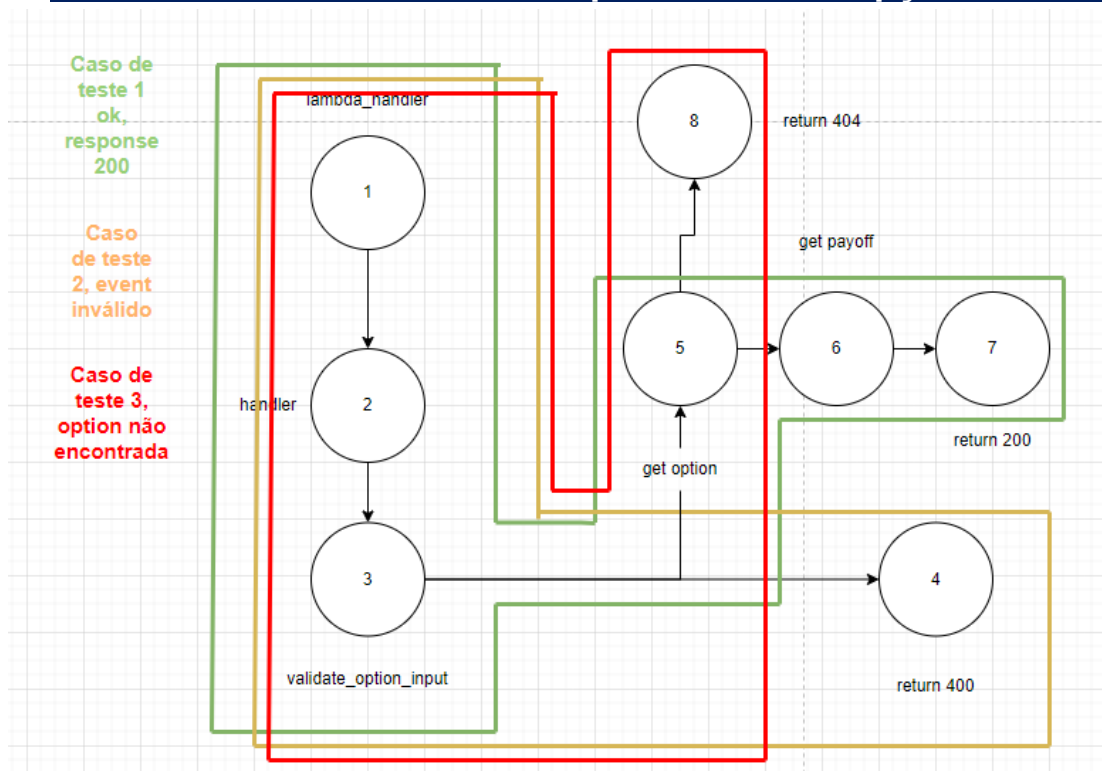


Figura 24 - Análise de todos os nós para busca de opção backend

## 8.3. 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	<pre>base_case = {   "event": {     "body": {"id": "60137665a3315885b579abe6803b55d0", "name": "BOVAA100"}   },   "context": None }</pre>	<pre>{   "statusCode": 200,   "body":   json.dumps(result) }</pre>
CT2	1-2,2-3,3-4	base_case={}	<pre>{   "statusCode": 400,</pre>

			"body": f"bad request, body not found at event"
CT3	1-2,2-3,3-5,5-8	<pre>base_case={   "event": {     "body": '{"id": "ABCD7665a3315885b579abe6803b55d0", "name": "BOVAA100"}'   },   "context": None } or base_case={   "event": {     "body": '{"id": "60137665a3315885b579abe6803b55d0", "name": "ABCDEFGH"}'   },   "context": None }</pre>	<pre>{   "statusCode": 404,   "body": f"Option not found!" }</pre>

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

## 8.4. 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

## 8.5. Função de Search

```

async def handler(event):
    results = mock
    return {
        "statusCode": 200,
        "body": json.dumps(results)
    }

```

Figura 25 - Função de Search

## 8.6. Análise de todos os nós para Search

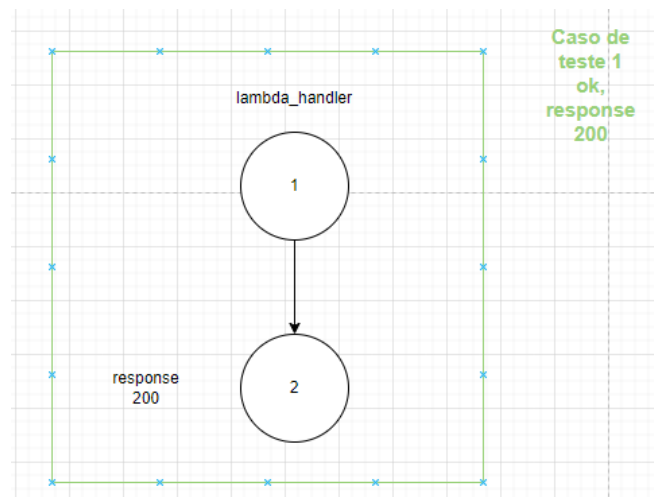


Figura 26 - Análise de todos os nós para Search

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

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

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

## 9. 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

### 9.1. Função Health

```
def lambda_handler(event, context):  
    return {  
        "statusCode": 200,  
        "body": json.dumps("Health Checked!")  
    }
```

Figura 28 - Função Health

### 9.2. 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.

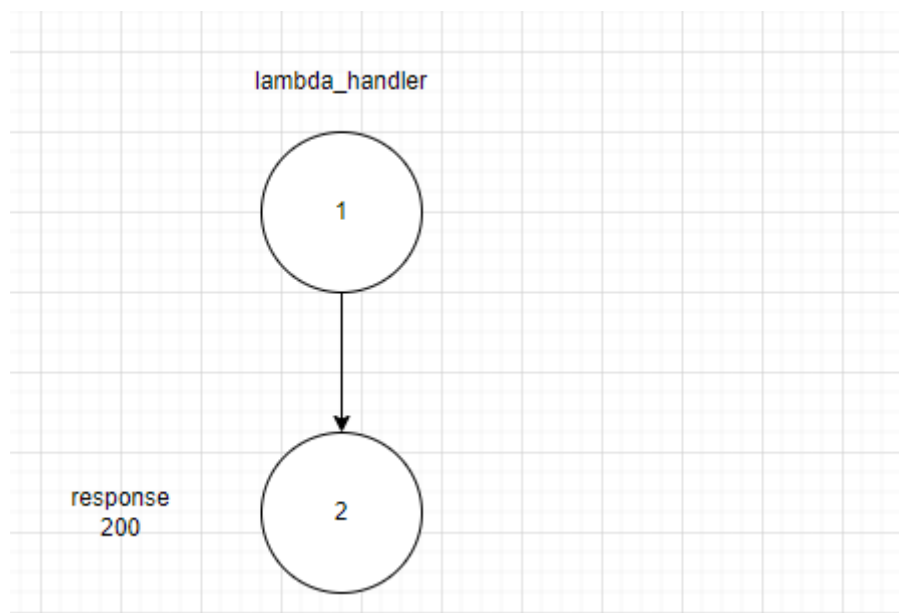


Figura 29 - GFC para cálculo de health check

### 9.3. 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.

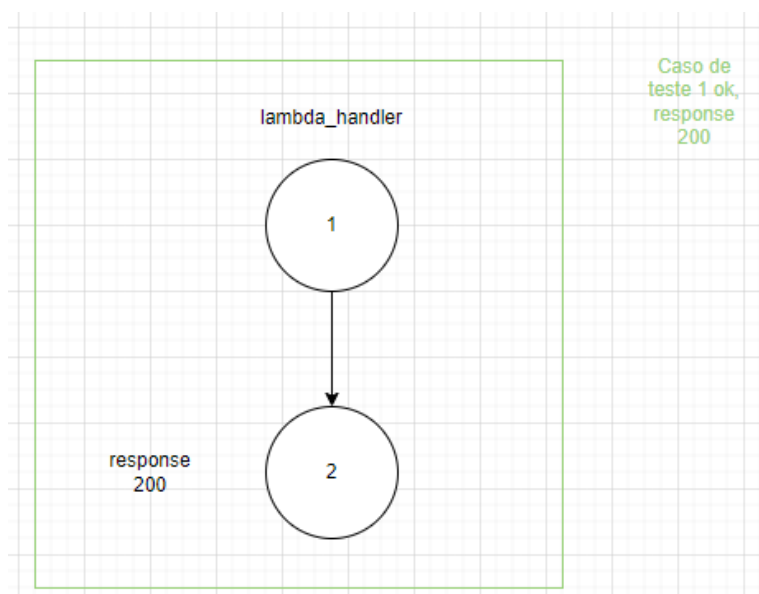


Figura 30 - Análise de todos os nós para validar o health check

### 9.4. Análise de todas as arestas para validar o health

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

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

## 10. 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.

### 10.1. 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:

#### 10.1.1. Função de validação de inserção

```
1  function validate(obj) {  
2      if(  
3          obj.name.length > 3 &&  
4          obj.exercise_price > 0 &&  
5          obj.close_price > 0 &&  
6          obj.contracts > 0  
7      ) return true;  
8      else return false;  
9  }
```

Figura 31 - Função de validação da inserção

#### 10.1.2. GFC para validação de inserção

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

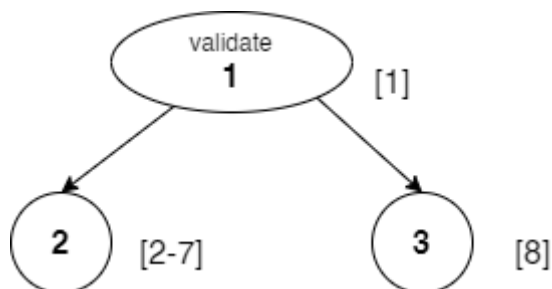


Figura 32 – CFG de validação de inserção

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

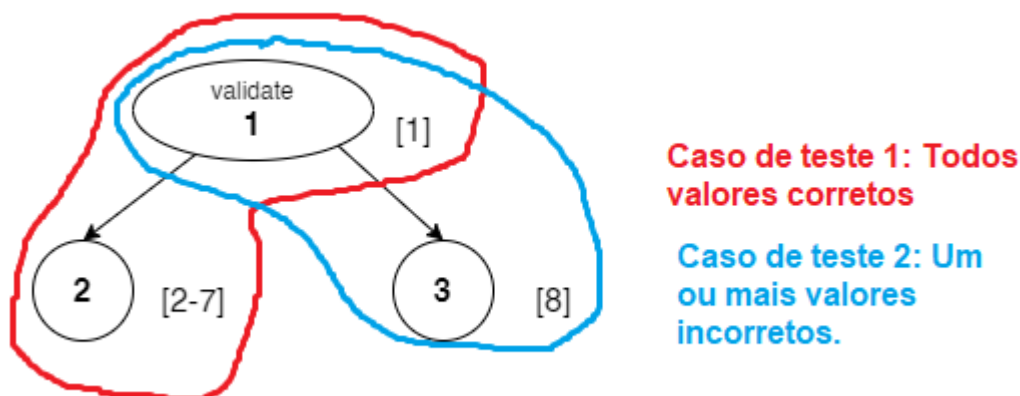


Figura 33 – Análise de todos os nós para validar inserção

### 10.1.4. 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