

Underlying

PROJETO 2022.01 - UNDERLYING
Testes Funcionais (DTF)
Caixa-Preta

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

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
20/06/2022	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
21/06/2022	0.1	Revisão do documento de teste	Lucas
17/07/2022	0.2	Revisão do documento de teste	Ivan



Underlying



Índice de Ilustrações

<i>Figura 1 - Cobertura de testes – Serviço de Estratégias</i>	<i>9</i>
<i>Figura 2 - Cobertura de testes – Serviço de Opções</i>	<i>10</i>
<i>Figura 3 - Cobertura de testes – Frontend</i>	<i>11</i>
<i>Figura 4 - Teste no VS Code do Create</i>	<i>37</i>
<i>Figura 5 - Teste no VS Code Update (Delete e Share)</i>	<i>47</i>
<i>Figura 6 - Teste no VS Code busca de opções</i>	<i>52</i>
<i>Figura 7 - Interface de busca de opções</i>	<i>55</i>
<i>Figura 8 - Tela de cadastro de usuário</i>	<i>58</i>
<i>Figura 9 - Tela de autenticação de usuário</i>	<i>60</i>



Underlying



ÍNDICE DE TABELA

<i>Tabela 1 - Responsabilidade dos testes funcionais</i>	<i>8</i>
<i>Tabela 2 - Testes da Função validate_item</i>	<i>33</i>
<i>Tabela 3 - Suíte de Testes combinatorial do Create</i>	<i>36</i>
<i>Tabela 4- Suíte de Testes da função shared</i>	<i>41</i>
<i>Tabela 5 -Suíte de Testes da Função Delete</i>	<i>44</i>
<i>Tabela 6 - Suíte de Testes combinatorial</i>	<i>46</i>
<i>Tabela 7 - Testes da Função Option</i>	<i>51</i>
<i>Tabela 8 - Testes Combinatorial da busca de Opções</i>	<i>52</i>
<i>Tabela 9 - Testes do retorno da busca de Opções</i>	<i>54</i>
<i>Tabela 10 - Testes de autenticação do usuário</i>	<i>63</i>



Underlying



Sumário

1. Introdução	7
1.1 Responsabilidades de testes da equipe de projeto	7
2. COBERTURA DOS TESTES	8
3. Teste Criação das estratégias de opções	11
3.1 Teste Função Validate_item	12
3.1.1 Restrições da Função item	12
3.1.2 Teste Funcional da Função item	14
3.1.3 Suíte de Testes da Função item	16
3.2 Teste Combinatorial do Create	33
3.2.1 Suíte de Testes Combinatorial do Create	33
4. Teste Update das estratégias de opções	37
4.1 Teste Função Share	38
4.1.1 Restrições da Função Share	38
4.1.2 Teste Funcional da Função Share	38
4.1.3 Suíte de Testes da Função Share	39
4.2 Teste Função Delete	41
4.2.1 Restrições da Função Delete	41
4.2.2 Teste Funcional da Função Delete	42
4.2.3 Suíte de Testes da Função Delete	42
4.3 Teste Combinatorial do Update	44
4.3.1 Suíte de Testes Combinatorial do Update	45
5. Teste buscar informações de opções	47
5.1 Teste Função Option	48
5.1.1 Restrições da Função Option	48
5.1.2 Teste Funcional da função Option	48
5.1.3 Suíte de Testes da Função Option	49
5.2 Teste Combinatorial da busca de opções	51
5.2.1 Suíte de Testes Combinatorial da busca de Opções	51
6. Teste de retorno da buscas das opções	52
6.1 Teste Funcional para retorno da busca de Opções	52
6.1.1 Suíte de Testes do retorno da busca de Opções	53
6.2 Teste Combinatorial do retorno da busca de opções	54
6.2.1 Suíte de Testes Combinatorial do retorno de busca de Opções	55
7. Teste do cadastro de usuário	58
7.1.1 Restrições dos campos de cadastro de usuário	58
7.1.2 Teste Combinatorial do cadastro de usuário	59
7.1.3 Suíte de Testes Combinatorial do cadastro de usuário	59



Underlying



8.	<i>Teste de autenticação do usuário</i>	60
8.1.1	Restrições dos campos de autenticação do usuário	60
8.1.2	Casos de testes de autenticação do usuário	61
8.1.3	Suíte de Testes de autenticação do usuário	62
8.1.4	Teste Combinatorial de autenticação do usuário	63
8.1.5	Suíte de Testes Combinatorial de autenticação do usuário	63
9.	<i>Teste de inserção de opção fictícia</i>	64
9.1.1	Restrições dos campos de inserção da opção fictícia:	64
9.1.2	Teste Combinatorial de inserção da opção fictícia	65
9.1.3	Suíte de Testes Combinatorial de inserção da opção fictícia	65

1. INTRODUÇÃO

Esse documento apresenta os testes funcionais que são fundamentais para garantir a qualidade do sistema. Uma vez que os testes funcionais permitem que os testes ocorram de uma forma mais eficiente e rápida, possibilitando encontrar as não conformidades do software em relação aos requisitos do sistema. Também são conhecidos como teste “caixa-preta”, estes são definidos de acordo com os requisitos funcionais do software. Como não há conhecimento sobre a operação interna do programa, o analista concentra-se nas funções que o software contemplará. Baseado na especificação determina-se as saídas que são esperadas para um determinado conjunto de dados.

Os testes funcionais revelam vários problemas como, por exemplo, funções incorretas ou omitidas; erros de interface; erros de comportamento ou desempenho; erros de iniciação e término, entre outros.

A principal técnica utilizada nos testes funcionais é a de particionamento de equivalência, que visa uma divisão em subconjuntos das entradas de valores de acordo com as funcionalidades do software, por exemplo, a inserção de uma massa dados para validar o processo de inserção. Seu princípio é a escolha da melhor abordagem a ser utilizada e a melhor maneira de se obter a validação dos erros e aumento da confiabilidade (BRUNELI, 2006). Uma maneira muito eficaz é analisar os dados resultantes e melhorar validações e verificações de entrada.

Para os testes funcionais deste projeto além da técnica de **particionamento de equivalência**, outra técnica que foi utilizada para garantir a qualidade do Underlying foram os testes de **Análise do valor limite e o testes combinatórios**.

1.1 Responsabilidades de testes da equipe de projeto

A seguir segue as reponsabilidades definidas para os testes Funcionais do sistema Underlying:

Testes Funcionais do Sistema Underlying		
Teste modulo	Descrição	responsável
Criar estratégia de opções (backend)	Realiza teste para o armazenar os dados de entrada referentes a uma estratégia de opções no banco de dados DynamoDB	Lucas
Update das estratégias de opções	Realiza o teste de atualização do compartilhamento e	Ivan

(backend)	remoção da estratégia de opções (backend) no banco DynamoDB	
Buscar informação de opções (backend)	Realiza os testes para buscar dados de uma opção já armazenada no S3 Bucket	Bruno Brandão
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 funcionais

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 (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 – X% de cobertura de código

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



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 3 - Cobertura de testes – Frontend

3. TESTE CRIAÇÃO DAS ESTRATÉGIAS DE OPÇÕES

A criação de estratégia de opções está contida em uma AWS Lambda e tem a função de armazenar os dados de entrada referentes a uma estratégia de opções no banco de dados não relacional DynamoDB, para isso ela deve normalizar os dados de entrada para que atenda aos requisitos e executar a validação dos campos, além de realizar a serialização dos dados antes do envio para a base de dados. Essa AWS lambda possui quatro funções principais:

1. **serialize_item**: responsável por serializar a estratégia no formato aceitável para inserção no banco de dados DynamoDB.
2. **save_item**: responsável por armazenar o item normalizado e serializado no DynamoDB, fazendo a atribuição do horário de inserção do item.
3. **validate_item**: responsável por validar a entrada do usuário garantindo que os requisitos dos campos sejam seguidos, e nenhum dado incorreto logicamente seja inserido na base de dados.
4. **lambda_handler**: responsável por receber os argumentos da chamada da API fazendo a chamada das funções necessárias para realizar o retorno adequado da criação da estratégia ao usuário.

3.1 Teste Função Validate_item

Considere a função `validate_item` (item) a ser codificada em Python. A função faz parte do módulo de criação de estratégia de opções do sistema Underlying. A função deve receber como parâmetro de entrada o dicionário referente a estratégia que o usuário deseja registrar. Esse dicionário deve conter o nome da estratégia, o username do usuário, o campo de compartilhamento da estratégia e o campo estratégia que se refere a um conjunto de opções, além disso, essa estratégia receberá um ID único ao passar pela função. Uma opção é um dicionário com os campos: nome, tipo, preço de exercício, tipo da transação, preço de fechamento, e número de contratos. Sendo assim, devem ser validados os campos da estratégia e os campos das opções que irão compor essa estratégia. Por se tratar de inserções em um banco de dados não relacional, a validação do tipo dos campos é essencial.

3.1.1 Restrições da Função item

A função deve validar os campos da estratégia de acordo com os seguintes critérios:

- Para que o **nome** seja válido deve:
 - ser do tipo string
 - ter entre 3 e 40 caracteres
- Para que o **username** seja válido deve:
 - ser do tipo string
 - ter entre 6 e 20 caracteres

Para que o **id** seja válido deve:

- ser do tipo string
- ser inicializado a partir de um UUID



Para que o **compartilhado** seja válido deve:

- ser do tipo booleano
- ser inicializado com valor False

Para que as **opções** sejam válidas devem:

- ser uma lista de opções
- atender os requisitos listados abaixo para o tipo opção

A função deve validar os campos de opção que compõe a estratégia de acordo com os seguintes critérios:

Para que o **nome** seja válido deve:

- ser do tipo string
- ter entre 3 e 40 caracteres

Para que o **tipo** seja válido deve:

- ser do tipo string
- assumir apenas um dos valores: "CALL" ou "PUT"

Para que o **preço de exercício** seja válido deve:

- ser do tipo float
- ser maior que zero

Para que o **tipo da transação** seja válido deve:

- ser do tipo string
- assumir apenas um dos valores: "LONG" ou "SHORT"

Para que o **preço de fechamento** seja válido deve:

- ser do tipo float
- ser maior que zero

Para que o campo **contratos** seja válido deve:

- ser do tipo inteiro
- ser maior do que zero

No caso de um ou mais campos inválidos, a API deve retornar o código de status 422 seguido da mensagem "Input field validation error" acompanhado do erro que foi gerado a partir da requisição. Em caso de sucesso, a API deve retornar o código de status 200 seguido do objeto da estratégia criada.



Underlying



3.1.2 Teste Funcional da Função item

A seguir ser apresentado os critérios funcionais sistemáticos para a realização dos testes da estratégia(A1) e das opções(A2):

A1 - **Critério Funcional Sistemático** - Classes de Equivalência (C) e Análise do Valor Limite para **estratégia**

- Comprimento do **nome**:
 - C1 – nome de 3 caracteres → válido
 - C2 – nome de 40 caracteres → válido
 - C3 – nome com 2 caracteres → inválido
 - C4 – nome com 41 caracteres → inválido
 - C5 – nome vazio → inválido
- Tipo do **nome**:
 - C6 – nome do tipo string → válido
 - C7 – nome do tipo inteiro → inválido
 - C8 – nome do tipo booleano → inválido
- Comprimento de **opções**:
 - C9 – opções com 1 item → válido
 - C10 – opções com 2 itens → válido
 - C11 – opções vazio → inválido
- Tipo de **opções**:
 - C12 – opções do tipo lista → válido
 - C13 – opções do tipo dicionário → inválido
 - C14 – opções do tipo null → inválido
- Comprimento do **username**:
 - C15 – username de 6 caracteres → válido
 - C16 – username de 20 caracteres → válido
 - C17 – username com 5 caracteres → inválido
 - C18 – username com 21 caracteres → inválido
 - C19 – username vazio → inválido
- Tipo do **username**:
 - C20 – username do tipo string → válido
 - C21 – username do tipo inteiro → inválido
 - C22 – username do tipo booleano → inválido



A2 - **Critério Funcional Sistemático** - Classes de Equivalência (C) e Análise do Valor Limite para **opção**:

- Comprimento do **nome**:
 - C23 – nome de 3 caracteres → válido
 - C24 – nome de 40 caracteres → válido
 - C25 – nome com 2 caracteres → inválido
 - C26 – nome com 41 caracteres → inválido
 - C27 – nome vazio → inválido
- Tipo do **nome**:
 - C28 – nome do tipo string → válido
 - C29 – nome do tipo inteiro → inválido
 - C30 – nome do tipo booleano → inválido
- Tipo do **tipo**:
 - C31 – tipo do tipo string → válido
 - C32 – tipo do tipo inteiro → inválido
 - C33 – tipo do tipo null → inválido
- Valores do **tipo**:
 - C34 – valor do tipo é CALL → válido
 - C35 – nome do tipo é PUT → válido
 - C36 – valor do tipo é call → válido
 - C37 – valor do tipo é long → inválido
- Tipo do **preço de exercício**:
 - C38 – tipo do preço de exercício float → válido
 - C39 – tipo do preço de exercício string → inválido
 - C40 – tipo do preço de exercício null → inválido
- Valores do **preço de exercício**:
 - C41 – valor do preço de exercício é 12,34 → válido
 - C42 – valor do preço de exercício é -12,34 → inválido
 - C43 – valor do preço de exercício é 0 → inválido
- Tipo do **tipo da transação**:
 - C44 – tipo do tipo da transação string → válido
 - C45 – tipo do tipo da transação float → inválido
 - C46 – tipo do tipo da transação null → inválido



- Valores do **tipo da transação**:
 - C47 – valor do tipo da transação é LONG → válido
 - C48 – nome do tipo da transação é SHORT → válido
 - C49 – valor do tipo da transação é short → válido
 - C50 – valor do tipo da transação é CALL → inválido
- Tipo do **preço de fechamento**:
 - C51 – tipo do preço de fechamento float → válido
 - C52 – tipo do preço de fechamento string → inválido
 - C53 – tipo do preço de fechamento null → inválido
- Valores do **preço de fechamento**:
 - C54 – valor do preço de fechamento é 11,24 → válido
 - C55 – valor do preço de fechamento é -11,24 → inválido
 - C56 – valor do preço de fechamento é 0 → inválido
- Tipo do **contratos**:
 - C57 – tipo do contratos inteiro → válido
 - C58 – tipo do contratos string → inválido
 - C59 – tipo do contratos null → inválido
- Valores do **contratos**:
 - C60 – valor do contratos é 100 → válido
 - C61 – valor do contratos é -34 → inválido
 - C62 – valor do contratos é 80.42 → inválido
 - C63 – valor do contratos é 0 → inválido

3.1.3 Suíte de Testes da Função item

A Tabela a seguir apresentará a realização de cada teste planejado da secção anterior:

Classe de Testes	Classes Relacionadas	Caso de Teste
CT1	C1	Obj = { "username": "blazzi", "name": "ABC", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, }]}

		<pre> "type": "PUT" }] } </pre> <p>Saída = Obj</p>
CT2	C2	<pre> Obj = { "username": "blazzi", "name": "Estratégia de validação Long Straddle 22", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = Obj</p>
CT3	C3	<pre> Obj = { "username": "blazzi", "name": "22", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT4	C4	<pre> Obj = { "username": "blazzi", "name": "Estratégia de validação Long Straddle 022", "strategy": [{ </pre>



Underlying

		<pre> "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT5	C5	<pre> Obj = { "username": "blazzi", "name": "", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT6	C6, C9, C12, C15, C20, C28, C31, C35, C38, C41, C44, C47, C51, C54, C57, C60	<pre> Obj = { "username": "blazzi", "name": "Estratégia de validação Long Straddle", "strategy": [{ "name": "CUSTS12", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída =</p>
CT7	C7	<pre> Obj = { "username": "blazzi", </pre>



Underlying

		<pre>"name": 22, "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT8	C8	<pre>Obj = { "username": "blazzi", "name": True, "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = Obj</p>
CT9	C10	<pre>Obj = { "username": "blazzi", "name": True, "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }, { "name": "CUSTB12 - SHORT", "exercise_price": 12.32, "transaction_type": "SHORT", </pre>



Underlying



		<pre> "close_price": 1.23, "contracts": 1, "type": "CALL" }] } </pre> <p>Saída = Obj</p>
CT10	C11	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [] } </pre> <p>Saída = ValidationError</p>
CT11	C13	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": { "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" } } </pre> <p>Saída = ValidationError</p>
CT12	C14	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": None } </pre> <p>Saída = ValidationError</p>
CT13	C16	<pre> Obj = { "username": "lucas_tiense_blazzi1", "name": "Estratégia Long Straddle", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, </pre>



Underlying

		<pre> "type": "PUT" }] } Saída = Obj</pre>
CT14	C17	<pre>Obj = { "username": "joao1", "name": "Estratégia Long Straddle", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT15	C18	<pre>Obj = { "username": "lucas_tiense_blazzi22", "name": "Estratégia Long Straddle", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT16	C19	<pre>Obj = { "username": "", "name": "Estratégia Long Straddle", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32,</pre>

		<pre> "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT17	C21	<pre> Obj = { "username": 22, "name": "Estratégia Long Straddle", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT18	C22	<pre> Obj = { "username": True, "name": "Estratégia Long Straddle", "strategy": [{ "name": "CUSTS12 - LONG", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT19	C23	<pre> Obj = { "username": "blazzi", "name": "Estratégia de validação Long Straddle", "strategy": [</pre>

		<pre>{ "name": "ITS", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }</pre> <p>Saída = ValidationError</p>
CT20	C24	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13 - Random Undelying - Exercise: 12", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = Obj</p>
CT21	C25	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RD", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT22	C26	<pre>Obj =</pre>

		<pre>{ "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13 - Random Undelying - Exercise: 12,89", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT23	C27	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT24	C29	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": 13, "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre>

		<pre>} Saída = ValidationError</pre>
CT25	C30	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": True, "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": True }] }</pre> <p>Saída = Obj</p>
CT26	C32	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": 1 }] }</pre> <p>Saída = ValidationError</p>
CT27	C33	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1,</pre>

		<pre> "type": None }] } </pre> <p>Saída = ValidationError</p>
CT28	C34	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "CALL" }] } </pre> <p>Saída = Obj</p>
CT29	C36	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "call" }] } </pre> <p>Saída = Obj</p>
CT30	C37	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.32, </pre>

		<pre> "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "long" }] } </pre> <p>Saída = ValidationError</p>
CT31	C39	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": "invalid", "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT32	C40	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": None, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT33	C42	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [</pre>



Underlying



		<pre>{ "name": "RDNS13", "exercise_price": -12.34, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }</pre> <p>Saída = ValidationError</p>
CT34	C43	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 0, "transaction_type": "LONG", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT35	C45	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": 123.41, "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT36	C46	<pre>Obj = {</pre>



Underlying



		<pre>"username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": None, "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT37	C48	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "SHORT", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre> <p>Saída = ValidationError</p>
CT38	C49	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "short", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre>



Underlying

		Saída = Obj
CT39	C50	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "CALL", "close_price": 1.23, "contracts": 1, "type": "PUT" }] }</pre>
		Saída = ValidationError
CT40	C52	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "LONG", "close_price": "invalid", "contracts": 1, "type": "PUT" }] }</pre>
		Saída = ValidationError
CT41	C53	<pre>Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "LONG", "close_price": None, "contracts": 1, "type": "PUT" }] }</pre>

		<pre>] } Saída = ValidationError </pre>
CT42	C55	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "LONG", "close_price": -11.24, "contracts": 1, "type": "PUT" }] } Saída = ValidationError </pre>
CT43	C56	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "LONG", "close_price": 0 "contracts": 1, "type": "PUT" }] } Saída = ValidationError </pre>
CT44	C58	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "LONG", "close_price": 11.24 }] } </pre>

		<pre> "contracts": "invalid", "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT45	C59	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "LONG", "close_price": 11.24 "contracts": None, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT46	C61	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "LONG", "close_price": 11.24 "contracts": -34, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>
CT47	C62	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", </pre>

		<pre> "exercise_price": 12.34, "transaction_type": "LONG", "close_price": 11.24 "contracts": 80.34, "type": "PUT" }] } </pre> <p>Saída = Obj</p>
CT48	C63	<pre> Obj = { "username": "blazzi", "name": "Estratégia Long Straddle", "strategy": [{ "name": "RDNS13", "exercise_price": 12.34, "transaction_type": "LONG", "close_price": 11.24 "contracts": 0, "type": "PUT" }] } </pre> <p>Saída = ValidationError</p>

Tabela 2 - Testes da Função validate_item

3.2 Teste Combinatorial do Create

Usando o critério de teste combinatorial, propor exemplos de casos de teste para o módulo de inserção de estratégias do backend – Registrar Estratégia.

Possíveis retornos de cada campo:

- **Nome:** válido / inválido
- **Username:** válido / inválido
- **Opções:** válido / inválido

Combinações de entradas: 2x2x2 = 8

3.2.1 Suíte de Testes Combinatorial do Create

A Tabela a seguir apresenta a realização dos testes combinatoriais para o módulo create.

Entrada Valida	Classe de teste
----------------	-----------------



Underlying



Nome: válido Username: válido Opções: válido	CT1: { "username": "blazzi", "name": "Estratégia de validação Long Straddle", "strategy": [{ "name": "CUSTS12", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 11.24, "contracts": 1, "type": "PUT" }] }
Entrada Invalida	Classe de teste
Nome: inválido Username: válido Opções: válido	CT2: { "username": "blazzi", "name": "Estratégia de validação Long Straddle – Underlying GBT – Exercise: 14,4", "strategy": [{ "name": "CUSTS12", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 11.24, "contracts": 1, "type": "PUT" }] }
Nome: inválido Username: inválido Opções: válido	CT3 { "username": "lucas_tiense_blazzi_2022", "name": "Estratégia de validação Long Straddle – Underlying GBT – Exercise: 14,4", "strategy": [{ "name": "CUSTS12", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 11.24, "contracts": 1, "type": "PUT" }] }



Underlying



Nome: inválido Username: válido Opções: inválido	CT4 { "username": "blazzi", "name": "Estratégia de validação Long Straddle – Underlying GBT – Exercise: 14,4", "strategy": [{ "name": "CUSTS12", "exercise_price": 12.32, "transaction_type": "PUT", "close_price": 11.24, "contracts": 1, "type": "PUT" }] }
Nome: inválido Username: inválido Opções: inválido	CT5 { "username": "lucas_tiense_blazzi_2022", "name": "Estratégia de validação Long Straddle – Underlying GBT – Exercise: 14,4", "strategy": [{ "name": "CUSTS12", "exercise_price": 12.32, "transaction_type": "PUT", "close_price": 11.24, "contracts": 1, "type": "PUT" }] }
Nome: válido Username: inválido Opções: válido	CT6 { "username": "lucas_tiense_blazzi_2022", "name": "Estratégia de validação Long Straddle", "strategy": [{ "name": "CUSTS12", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 11.24, "contracts": 1, "type": "PUT" }] }
Nome: válido	CT7

Username: válido Opções: inválido	<pre>{ "username": "blazzi", "name": "Estratégia de validação Long Straddle", "strategy": [{ "name": "CUSTS12", "exercise_price": 12.32, "transaction_type": "LONG", "close_price": 11.24, "contracts": 1, "type": "long" }] }</pre>
Nome: válido Username: inválido Opções: inválido	<pre>CT8 { "username": "lu", "name": "Estratégia de validação Long Straddle", "strategy": [{ "name": "CUSTS12", "exercise_price": 0, "transaction_type": "LONG", "close_price": 11.24, "contracts": 1, "type": "PUT" }] }</pre>

Tabela 3 - Suíte de Testes combinatorial do Create

A seguir segue um print, a exemplo, sobre a saída do teste em Python no VS Code:



Underlying



```
----- coverage: platform win32, python 3.9.4-final-0 -----
Name                                                    Stmts  Miss  Cover
-----
strategies\functions\api\create\__init__.py              0      0  100%
strategies\functions\api\create\app\__init__.py          0      0  100%
strategies\functions\api\create\app\app.py              31     15   52%
strategies\functions\api\create\app\schema.py            28      0  100%
strategies\functions\api\create\tests\__init__.py         0      0  100%
strategies\functions\api\create\tests\input.py           348     0  100%
strategies\functions\api\create\tests\test_lambda_handler.py 0      0  100%
strategies\functions\api\create\tests\test_validate_item.py 254     0  100%
strategies\functions\api\update\__init__.py              0      0  100%
strategies\functions\api\update\app\__init__.py          0      0  100%
strategies\functions\api\update\app\app.py              74     39   47%
strategies\functions\api\update\app\schema.py            28      6   79%
strategies\functions\api\update\tests\__init__.py         0      0  100%
strategies\functions\api\update\tests\input.py           12      0  100%
strategies\functions\api\update\tests\test_validate_share.py 16      0  100%
strategies\test_env\aws.py                               30      8   73%
strategies\test_env\data.py                              10      0  100%
strategies\test_env\resources.py                         10      0  100%
strategies\test_env\test_resources.py                   20      0  100%
-----
TOTAL                                                    861     68   92%

===== 75 passed in 11.15s =====
(coincede-venv) D:\Documents\GitHub\underlying-project>
```

Figura 4 - Teste no VS Code do Create

4. TESTE UPDATE DAS ESTRATÉGIAS DE OPÇÕES

A atualização da estratégia de opções está contida em uma AWS Lambda e tem a função de atualizar os dados de entrada referentes a uma estratégia de opções no banco de dados não relacional DynamoDB, para isso ela deve normalizar os dados de entrada para que atenda aos requisitos e executar a validação dos campos, além de realizar a serialização dos dados antes do envio para a base de dados. Essa AWS lambda possui duas funções principais para a atualização das estratégias, elas são:

- 1- SHARE: Realiza a atualização do compartilhamento da estratégia.** A função possui dois campos de entrada: O id que identifica a estratégia através de uma string alfanumérica e o shared que é do tipo booleano e assume verdadeiro ou falso. Quando verdadeiro indicara que a estrutura com o id informado será compartilhada, quando falso indica que a mesma deixará de ser compartilhada.
- 2- Delete: Remove a estratégia criada pelo usuário.** A função possui dois campos de entrada: O id que identifica a estratégia através de uma string alfanumérica e o deleted que é do tipo booleano e assume verdadeiro ou falso. Quando verdadeiro indicara que a estrutura com o id informado será



Underlying



removida, quando falso indica que a mesma será removida, não podendo assim mais ser visualizada.

4.1 Teste Função Share

O teste unitário da função share consistirá na validação através da função `test_validate_share.py` (item) a ser codificada em Python que faz parte do módulo de atualização de estratégia de opções do sistema Underlying. A função deve receber como parâmetro de entrada o dicionário referente a atualização do compartilhamento da estratégia que o usuário deseja realizar. Esse dicionário deve conter o ID único da estratégia e o campo booleano de compartilhamento da estratégia. Portanto uma atualização do compartilhamento da estratégia será um dicionário com os campos: ID e Shared. Sendo assim, devem ser validados os campos da atualização da estratégia, uma vez que a atualização será feita em um banco de dados não relacional, a validação desses campos é essencial para garantir a qualidade do sistema Underlying.

4.1.1 Restrições da Função Share

A função shared apresenta os campos id e shared como parâmetro de entrada e possui as seguintes restrições para ser válido:

- Campo id deve ser valido com:
 - 36 caracteres de tamanho
 - Ser do tipo string (alfanumericos)
- Campo shared deve ser valido com:
 - Ser do tipo booleano (assumir valores: Verdadeiro ou Falso)

No caso de um ou mais campos inválidos, a API deve retornar o código de status 404 seguido da mensagem "Option not found!" acompanhado do erro que foi gerado a partir da requisição. Em caso de sucesso, a API deve retornar o código de status 200 seguido do objeto da estratégia criada.

4.1.2 Teste Funcional da Função Share

O teste funcional da função share será representado por A1. Será realizado o Critério Funcional Sistemático com Classes de Equivalência (C) e análise do Valor Limite para a unidade Update. Os testes a serem realizado serão apresentados a seguir:

- **Comprimento do id:**
 - C1 – id de 36 caracteres → válido

- C2 – id de 35 caracteres → inválido
 - C3 – id com 37 caracteres → inválido
 - C4 – id com 1 caracteres → inválido
 - C5 – id vazio → inválido
- **Tipo do id:**
 - C6 – id do tipo string → válido
 - C7 – id do tipo inteiro → inválido
 - C8 – id do tipo booleano → inválido
 - **Tipo do shared:**
 - C9 – shared do tipo booleano → válido
 - C10 – shared do tipo inteiro → inválido
 - C11 – shared do tipo string → inválido
 - C12 – shared do tipo vazio → inválido

4.1.3 Suíte de Testes da Função Share

A Tabela a seguir apresentará a realização de cada teste planejado da secção anterior:

Comprimento do id		
Id	Teste a validar	Código das entradas e saída validada
C1	Id de 36 caracteres → válido	obj= { "id":"d5c78622-cc45-4c20-9ada-a34cf39234e1", "shared": False } Saída = objeto atualizado/descompartilhado
C2	id de 35 caracteres→ inválido	obj= { "id":" d5c78622-cc45-4c20-9ada-a34cf39234e ", //tirando o 1 "shared": False } Saída = ValidationError
C3	id de 37 caracteres→ inválido	obj= { "id":"d5c78622-cc45-4c20-9ada-a34cf39234e12", //acrescentando 2 "shared": False } Saída = ValidationError
C4	id de 1 caracteres→ inválido	obj= {

		<pre>"id": "d", "shared": False } Saída = ValidationError</pre>
C5	id vazio → inválido	<pre>obj= { "id": "", "shared": False } Saída = ValidationError</pre>
Tipo ID		
Id	Teste a validar	Código das entradas e saída validada
C6	id do tipo string → válido	<pre>obj= { "id": "5dc78622-cc45-4c20-9ada- a34cf392341e", "shared": False } Saída = objeto atualizado/descompartilhado</pre>
C7	id do tipo inteiro → inválido	<pre>obj= { "id": 10, "shared": False } Saída = ValidationError</pre>
C8	id do tipo booleano → inválido	<pre>obj= { "id": False, "shared": False } Saída = ValidationError</pre>
C9	id do tipo float → inválido	<pre>obj= { "id": 2.134, "shared": False } Saída = ValidationError</pre>
Tipo do shared		
Id	Teste a validar	Código das entradas e saída validada
C10	shared do tipo booleano → válido	<pre>obj= { "id": "d5c78622-cc45-4c20-9ada- a34cf39234e1", "shared": True } Saída = objeto atualizado/compartilhado</pre>

C11	shared do tipo inteiro → inválido	obj= { "id":"d5c78622-cc45-4c20-9ada-a34cf39234e1", "shared": 50 } Saída = ValidationError
C12	shared do tipo string → inválido	obj= { "id":"d5c78622-cc45-4c20-9ada-a34cf39234e1", "shared": "teste" } Saída = ValidationError
C13	shared do tipo vazio → inválido	obj= { "id":"d5c78622-cc45-4c20-9ada-a34cf39234e1", "shared": "" } Saída = ValidationError

Tabela 4- Suíte de Testes da função shared

4.2 Teste Função Delete

O teste unitário da função delete consistirá na validação através da função `test_validate_delete.py` (item) a ser codificada em Python que faz parte do módulo de remoção da estratégia de opções do sistema Underlying. A função deve receber como parâmetro de entrada o dicionário referente a remoção da estratégia que o usuário deseja excluir. Esse dicionário deve conter o ID único da estratégia e o campo booleano para a remoção da estratégia. Portanto a exclusão da estratégia será um dicionário com os campos: ID e deleted. Sendo assim, devem ser validados os campos da remoção da estratégia, uma vez que a exclusão será feita em um banco de dados não relacional, a validação desses campos é essencial para garantir a qualidade do sistema Underlying.

4.2.1 Restrições da Função Delete

A função delete apresenta os campos `id` e `deleted` como parâmetro de entrada e possui as seguintes restrições para ser valido:

- Campo `id` deve ser valido com:
 - 36 caracteres de tamanho
 - Ser do tipo string (alfanumericos)
- Campo `deleted` deve ser valido com:
 - Ser do tipo booleano (assumir valores: Verdadeiro ou Falso)

4.2.2 Teste Funcional da Função Delete

O teste funcional da função delete será representado por A2. Será realizado o Critério Funcional Sistemático com Classes de Equivalência (C) e análise do Valor Limite para a unidade Update. Os testes a serem realizados serão apresentados a seguir:

- **Comprimento do id:**
 - C14 – id de 36 caracteres → válido
 - C15 – id de 35 caracteres → inválido
 - C16 – id com 37 caracteres → inválido
 - C17 – id com 1 caracteres → inválido
 - C18 – id vazio → inválido
- **Tipo do id:**
 - C19 – id do tipo string → válido
 - C20 – id do tipo inteiro → inválido
 - C21 – id do tipo booleano → inválido
- **Tipo do deleted:**
 - C22 – deleted do tipo booleano → válido
 - C23 – deleted do tipo inteiro → inválido
 - C25 – deleted do tipo string → inválido
 - C26 – deleted do tipo vazio → inválido

4.2.3 Suíte de Testes da Função Delete

A Tabela 02 a seguir apresenta a realização de cada teste planejado na seção 3.2:

Comprimento do id		
Id	Teste a validar	Código das entradas e saída validada
C14	Id de 36 caracteres → válido	<pre>obj= { "id":"d5c78622-cc45-4c20-9ada-a34cf39234e1", "deleted": False } Saída = objeto deletado</pre>
C15	id de 35 caracteres → inválido	<pre>obj= { "id":"d5c78622-cc45-4c20-9ada-a34cf39234e", //tirando o 1 "deleted": False }</pre>

		Saída = ValidationError
C16	id de 37 caracteres → inválido	obj= { "id": "d5c78622-cc45-4c20-9ada-a34cf39234e12", //acrescentando 2 "deleted": False } Saída = ValidationError
C17	id de 1 caracteres → inválido	obj= { "id": "d", "deleted": False } Saída = ValidationError
C18	id vazio → inválido	obj= { "id": "", "deleted": False } Saída = ValidationError
Tipo ID		
Id	Teste a validar	Código das entradas e saída validada
C19	id do tipo string → válido	obj= { "id": "5dc78622-cc45-4c20-9ada-a34cf392341e", "deleted": False } Saída = objeto deletado
C20	id do tipo inteiro → inválido	obj= { "id": 10, "deleted": False } Saída = ValidationError
C21	id do tipo booleano → inválido	obj= { "id": False, "deleted": False } Saída = ValidationError
C22	id do tipo float → inválido	obj= { "id": 2.134, "deleted": False } Saída = ValidationError
Tipo do deleted		
Id	Teste a validar	Código da entradas e saída validada

C23	deleted do tipo booleano → válido	obj= { "id": "d5c78622-cc45-4c20-9ada-a34cf39234e1", "deleted": True } Saída = objeto deletado
C24	deleted do tipo inteiro → inválido	obj= { "id": "d5c78622-cc45-4c20-9ada-a34cf39234e1", "deleted": 50 } Saída = ValidationError
C25	deleted do tipo string → inválido	obj= { "id": "d5c78622-cc45-4c20-9ada-a34cf39234e1", "deleted": "teste" } Saída = ValidationError
C26	deleted do tipo vazio → inválido	obj= { "id": "d5c78622-cc45-4c20-9ada-a34cf39234e1", "deleted": "" } Saída = ValidationError

Tabela 5 -Suíte de Testes da Função Delete

4.3 Teste Combinatorial do Update

Usando o critério de teste combinatorial, será proposto os casos de teste para o módulo Update compostos pelas duas funções principais de shared e delete, responsáveis respectivamente pela atualização e remoção da estratégia no backend.

Possíveis combinações com os campos:

- Função shared:
 - Id_(shared): válido / inválido
 - shared: válido / inválido
- ✓ Total de combinações de entradas: 2x2=4

- Função deleted:
 - Id_(deleted): válido / inválido
 - deleted: válido/inválido
- ✓ Total de combinações de entradas: 2x2=4

4.3.1 Suíte de Testes Combinatorial do Update

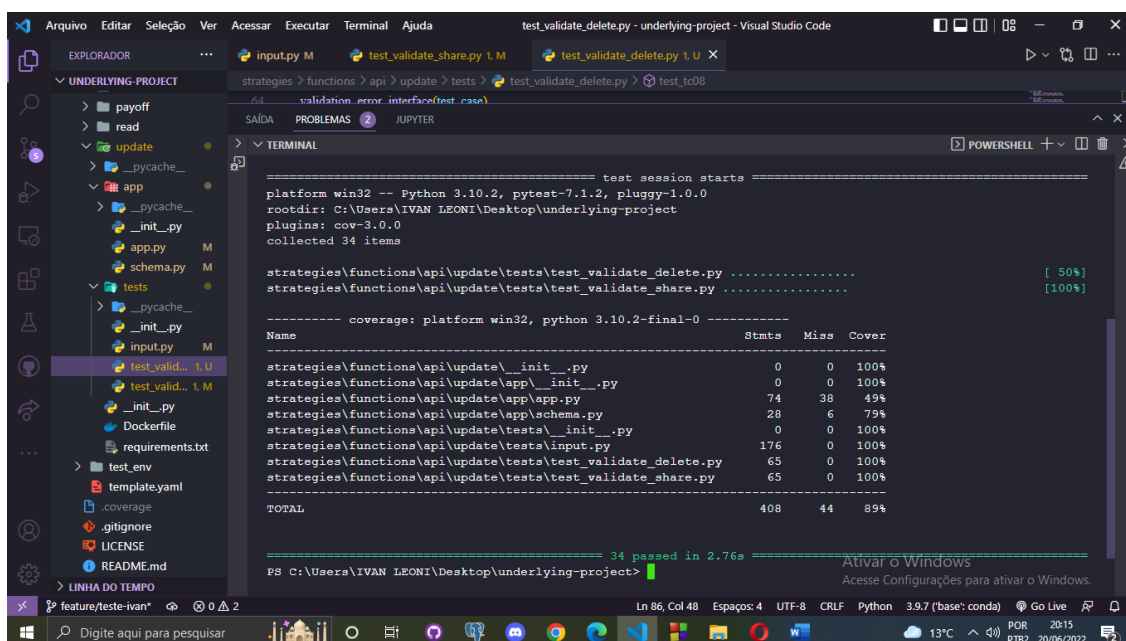
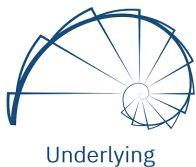
A Tabela a seguir apresenta a realização dos 8 testes combinatoriais para o módulo Update.

ENTRADAS VALIDAS		
Id	Teste	Código das entradas e saída validada
TC01	Id Valido, Shared Valido	Obj = { "id": "cc45-d5c78622-a34cf39234e1-4c20-9ada", "shared": True } Saída = objeto deletado
TC02	Id Valido, deleted Valido	Obj = { "id": "4c20-d5c78622-9ada-a34cf39234e1-cc45", "deleted": True } Saída = objeto atualizado/compartilhado
ENTRADAS INVALIDAS		
Id	Teste	Código das entradas e saída validada
TC03	Id Valido, Shared Invalido	Obj = { "id": "cc45-d5c78622-a34cf39234e1-4c20-9ada", "shared": "Azul" } Saída = ValidationError
TC04	Id Invalido, Shared valido	Obj = { "id": 43001 "shared": True }

		} Saída = ValidationError
TC05	Id Invalido, Shared Invalido	Obj = { "id": "sim" "shared": "sim" } Saída: ValidationError
TC06	Id Valido, Deleted Invalido	Obj = { "id": "4c20-d5c78622-9ada-a34cf39234e1-cc45", "deleted": "ok" } Saída: ValidationError
TC07	Id Invalido, Deleted Valido	Obj = { "id": " 5\$8ierFF", "deleted": True } Saída: ValidationError
TC08	Id Invalido, Deleted Invalido	Obj = { "id": " no", "deleted": "no" } Saída: ValidationError

Tabela 6 - Suíte de Testes combinatorial

A seguir segue um print, a exemplo, sobre a saída do teste em Python no VS Code:



```
test session starts
platform win32 -- Python 3.10.2, pytest-7.1.2, pluggy-1.0.0
rootdir: C:\Users\IVAN LEONI\Desktop\underlying-project
plugins: cov-3.0.0
collected 34 items

strategies\functions\api\update\tests\test_validate_delete.py ..... [ 50%]
strategies\functions\api\update\tests\test_validate_share.py ..... [100%]

----- coverage: platform win32, python 3.10.2-final-0 -----
Name                                                              Stmts   Miss  Cover
-----
strategies\functions\api\update\__init__.py                       0     0   100%
strategies\functions\api\update\app\__init__.py                   0     0   100%
strategies\functions\api\update\app\app.py                        74    38    49%
strategies\functions\api\update\app\schema.py                     28     6    79%
strategies\functions\api\update\tests\__init__.py                  0     0   100%
strategies\functions\api\update\tests\input.py                   176     0   100%
strategies\functions\api\update\tests\test_validate_delete.py      65     0   100%
strategies\functions\api\update\tests\test_validate_share.py      65     0   100%
TOTAL                                                            408    44    89%

34 passed in 2.76s
```

Figura 5 - Teste no VS Code Update (Delete e Share)

5. TESTE BUSCAR INFORMAÇÕES DE OPÇÕES

A busca de uma opção está contida em uma AWS Lambda e tem a função buscar dados de uma opção já armazenada no S3 Bucket, sendo a busca feita a partir do Nome e ID. É necessário que os dados de entrada referentes a uma opção sejam normalizados, para que a busca seja realizada de acordo com o seu padrão de armazenamento: BUCKET / NOME ARQUIVO / ID. Essa AWS lambda possui **cinco** funções principais:

1. **validate_option_input**: responsável por validar a entrada, garantindo que os requisitos dos campos sejam seguidos, e nenhum dado incorreto logicamente seja utilizado na busca de opções.
2. **get_option**: responsável por buscar no S3 a opção de acordo com a entrada.
3. **preprocess_payload**: responsável por normalizar as informações da Opção.
4. **get_payoff**: trazer informações de Payoff da opção.
5. **lambda_handler**: responsável por receber os argumentos da chamada da API fazendo a chamada das funções necessárias para realizar o retorno adequado da criação da estratégia ao usuário.



Underlying



5.1 Teste Função Option

Considere a função `validate_option_input(event)` codificada em Python. A função faz parte do módulo de busca de dados de uma opção do sistema Underlying. A função deve receber como parâmetro de entrada o dicionário referente a opção que o usuário deseja buscar. Esse dicionário deve conter o nome da opção e seu ID. Uma opção é um dicionário com os campos: nome, tipo, preço de exercício, tipo da transação, preço de fechamento, entre outros.

5.1.1 Restrições da Função Option

A função deve validar a busca da opção de acordo com os seguintes critérios:

- Para que o nome seja válido deve:
 - ser do tipo string
 - ter entre 5 e 8 caracteres
- Para que o id seja válido deve:
 - ser do tipo string
 - ter 32 caracteres

No caso de um ou mais campos inválidos, a API deve retornar o código de status 404 seguido da mensagem “Option not found!” acompanhado do erro que foi gerado a partir da requisição. Em caso de sucesso, a API deve retornar o código de status 200 seguido do objeto da estratégia criada.

5.1.2 Teste Funcional da função Option

A seguir ser apresentado os critérios funcionais sistemáticos para a realização dos testes das buscas de opções(A):

A1 - **Critério Funcional Sistemático** - Classes de Equivalência (C) e Análise do Valor Limite para a **Opção**

- Comprimento do **nome**:
 - o C1 – nome de 5 caracteres → válido
 - o C2 – nome de 8 caracteres → válido
 - o C3 – nome com 2 caracteres → inválido
 - o C4 – nome com 10 caracteres → inválido
 - o C5 – nome vazio → inválido
- Tipo do **nome**:
 - o C6 – nome do tipo string → válido
 - o C7 – nome do tipo inteiro → inválido
 - o C8 – nome do tipo booleano → inválido

- Comprimento do **id**:
 - o C9 – id de 32 caracteres → válido
 - o C10 – id de 31 caracteres → inválido
 - o C11 – id de 33 caracteres → inválido
- Tipo do **id**:
 - o C12 – id do tipo string → válido
 - o C13 – id do tipo dicionário → inválido
 - o C14 – id do tipo booleano → inválido

5.1.3 Suíte de Testes da Função Option

A Tabela a seguir apresentará a realização de cada teste planejado da secção anterior:

Classe de Testes	Classe Relacionadas	Caso de Teste
CT1	C1	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'ABCDE' } Saída = Obj
CT2	C2	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'BOVAA100' } Saída = Obj
CT3	C3	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'AB' } Saída = Obj
CT4	C4	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'ABCDEFGHIJ' } Saída = Obj
CT5	C5	Obj = { 'id': '60137665a3315885b579abe6803b55d0',



Underlying



		'name': "" } Saída = Obj
CT6	C6	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'BOVAA100' } Saída = Obj
CT7	C7	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': 22 } Saída = Obj
CT8	C8	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': True } Saída = Obj
CT9	C9	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'BOVAA100' } Saída = Obj
CT10	C10	Obj = { 'id': 'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDE', 'name': 'BOVAA100' } Saída = Obj
CT11	C11	Obj = { 'id': 'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFG', 'name': 'BOVAA100' } Saída = Obj
CT12	C12	Obj = { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'BOVAA100' } Saída = Obj
CT13	C13	Obj = { 'id': {'foo': 'bar'}, 'name': 'BOVAA100'

		<pre> } Saída = Obj </pre>
CT14	C14	<pre> Obj = { 'id': False, 'name': 'BOVAA100' } Saída = Obj </pre>

Tabela 7 - Testes da Função Option

5.2 Teste Combinatorial da busca de opções

Usando o critério de teste combinatorial, propor exemplos de casos de teste para o módulo de inserção de estratégias do backend – Buscar Dados de Opção.

Possíveis retornos de cada campo:

- Nome: válido / inválido
- ID: válido / inválido

Combinações de entradas: $2 \times 2 = 4$

5.2.1 Suíte de Testes Combinatorial da busca de Opções

A Tabela a seguir apresenta a realização dos 4 testes combinatoriais para o módulo de busca de opções:

Nome: válido ID: válido	CT1 { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'BOVAA100' }
Nome: inválido ID: válido	CT2 { 'id': '60137665a3315885b579abe6803b55d0', 'name': 'ABC' }
Nome: válido ID: inválido	CT3 { 'id': 'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDE', 'name': 'BOVAA100' }
Nome: inválido ID: inválido	CT4 { 'id': 'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFG', 'name': 'ABCDEFGHJIJ' }



Underlying



	}
--	---

Tabela 8 - Testes Combinatorial da busca de Opções

A seguir segue um print, a exemplo, sobre a saída do teste em Python no VS Code:

```
platform linux -- Python 3.8.0, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/bruno/Documents/unifei/develop/underlying-project/options/functions/api/info/test
plugins: asyncio-0.18.3
asyncio: mode=legacy
collected 18 items

test_get_option.py ..... [100%]

===== 18 passed, 1 warning in 1.54s =====
```

Figura 6 - Teste no VS Code busca de opções

6. TESTE DE RETORNO DA BUSCAS DAS OPÇÕES

Considere a função `validate_searchReturn(search)` em Javascript. A função faz parte da busca de opções do sistema *Underlying*. A função recebe uma lista de opções de uma busca e valida os seguintes requisitos:

- Nome de 8 caracteres da opção
- Nome de 5 caracteres do ativo base
- Tipo da opção é "CALL" ou "PUT"
- Data de lançamento é igual ou anterior a data atual e no formato "YYYY-MM-DD"
- Data de vencimento é maior que a data de lançamento e no formato "YYYY-MM-DD"
- Preço de fechamento é do tipo float
- Preço de exercício é do tipo float

No caso de erro de resultado da busca o sistema deve retornar *false*. E em caso de sucesso a função deve retornar a lista de opções.

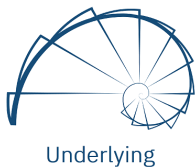
6.1 Teste Funcional para retorno da busca de Opções

A seguir ser apresentado os critérios funcionais sistemáticos para a realização dos testes do retorno da busca de opções (A):

A - Critério Funcional Sistemático - Classes de Equivalência (C) e Análise do Valor Limite para retorno da busca.

Nome de 8 caracteres da opção:

- C1: Nome de 8 caracteres (string) → válido
- C2: Nome do tipo inteiro → inválido
- C3: Nome do tipo booleano → inválido



Nome de 5 caracteres do ativo base

- C4:Nome de 5 caracteres (string)→ válido
- C5:Nome do tipo inteiro → inválido
- C6:Nome do tipo booleano → inválido

Tipo da opção

- C7:Tipo “CALL” ou “PUT”→ válido
- C8:Tipo “call” → válido
- C9:Tipo do tipo long→ inválido
- C10:Tipo do tipo null→ inválido

Data de lançamento

- C11:Data “2022-05-20” (Data atual = 2022-06-20)→ válido
- C12:Data “2022-07-20” (Data atual = 2022-06-20)→ inválido
- C13:Data “20-06-22” → inválido
- C14:Data do tipo null→ inválido

Data de vencimento

- C15:Data “2022-06-20” (Data de lançamento = 2022-05-20)→ válido
- C16:Data “2022-05-20” (Data de lançamento = 2022-06-20)→ inválido
- C17:Data “20-06-22” → inválido
- C18:Data do tipo→ inválido

Preço de fechamento

- C19:Preço float “12,34” → válido
- C20:Preço float “-12,34” → inválido
- C21:Preço null→ inválido

Preço de exercício

- C22:Preço float “12,34” → válido
- C23:Preço float “-12,34” → inválido
- C24:Preço null→ inválido

6.1.1 Suíte de Testes do retorno da busca de Opções

A Tabela a seguir apresentará a realização de cada teste planejado da secção anterior:



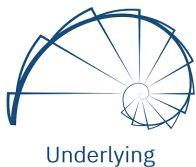
Underlying

Classes de teste	Classes relacionadas	Caso de teste
CT1	C1,C4,C7,C11, C15,C19,C22	lista = [name: “AAAAA111” , base: “AAAA” ,type: “CALL” , date: “2022-05-20” , date_expire: “2022-06-20” , close_price: 10.11 , exercise_price: 12.34] ; saida = lista
CT2	C2	lista = [name: “5555” , base: “AAAA” ,type: “CALL” , date: “2022-05-20” , date_expire: “2022-06-20” , close_price: 10.11 , exercise_price: 12.34] ; false
CT3	C6	lista = [name: “AAAAA111” , base: “true” ,type: “CALL” , date: “2022-05-20” , date_expire: “2022-06-20” , close_price: 10.11 , exercise_price: 12.34] ; false
CT4	C8	lista = [name: “AAAAA111” , base: “AAAA” ,type: “call” , date: “2022-05-20” , date_expire: “2022-06-20” , close_price: 10.11 , exercise_price: 12.34] ; saida = lista
CT5	C9	lista = [name: “AAAAA111” , base: “AAAA” ,type: “9007199254740991” , date: “2022-05-20” , date_expire: “2022-06-20” , close_price: 10.11 , exercise_price: 12.34] ; false
CT6	C14	lista = [name: “AAAAA111” , base: “AAAA” ,type: “CALL” , date: null , date_expire: “2022-06-20” , close_price: 10.11 , exercise_price: 12.34] ; false
CT7	C17	lista = [name: “AAAAA111” , base: “AAAA” ,type: “CALL” , date: “2022-05-20” , date_expire: “2022-03-20” , close_price: 10.11 , exercise_price: 12.34] ; false
CT8	C20	lista = [name: “AAAAA111” , base: “AAAA” ,type: “CALL” , date: “2022-05-20” , date_expire: “2022-06-20” , close_price: -10.11 , exercise_price: 12.34] ; false

Tabela 9 - Testes do retorno da busca de Opções

6.2 Teste Combinatorial do retorno da busca de opções

Usando o critério de teste combinatorial, propor exemplos de casos de teste para o retorno da interface de usuário (tela) Busca de Opções.



Underlying



Buscar Opções:

ITSAA110 ITSA4 CALL Data: 2022-01-14 Vencimento: 2022-01-21	Preço de fechamento: 0.01 Preço de exercício: 10.35
ITSAA114 ITSA4 CALL Data: 2022-01-14 Vencimento: 2022-01-21	Preço de fechamento: 0.01 Preço de exercício: 10.59

Figura 7 - Interface de busca de opções

Possíveis retornos de cada campo:

- Nome:** válido/inválido
- Nome ativo base:** válido/inválido
- Tipo:** válido/inválido
- Data de lançamento:** válido/inválido
- Data de vencimento:** válido/inválido
- Preço de fechamento:** válido/inválido
- Preço de exercício:** válido/inválido

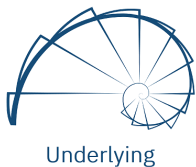
Combinações de retorno: $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^7 = 128$

6.2.1 Suíte de Testes Combinatorial do retorno de busca de Opções

A seguir apresentado a realização dos testes combinatoriais para o retorno de busca de opções:

Entradas válidas:

CT1: name: "AAAAA111" , válido
base: "AAAA" , válido
type: "CALL" , válido
date: "2022-05-20" , válido



Underlying



date_expire: "2022-06-20" , válido

close_price: 10.11 , válido

exercise_price: 12.34, válido

Entradas inválidas:

CT2: name: "12345678" , inválido

base: "AAAA" , válido

type: "CALL" , válido

date: "2022-05-20" , válido

date_expire: "2022-06-20" , válido

close_price: 10.11 , válido

exercise_price: 12.34, válido

CT3: name: "12345678" , inválido

base: false , inválido

type: "CALL" , válido

date: "2022-05-20" , válido

date_expire: "2022-06-20" , válido

close_price: 10.11 , válido

exercise_price: 12.34, válido

CT4: name: "12345678" , inválido

base: false , inválido

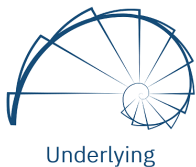
type: "9007199254740991" , inválido

date: "2022-05-20" , válido

date_expire: "2022-06-20" , válido

close_price: 10.11 , válido

exercise_price: 12.34, válido



Underlying



CT5: name: "12345678" , inválido
base: false , inválido
type: "9007199254740991" , inválido
date: null , inválido
date_expire: "2022-06-20" , válido
close_price: 10.11 , válido
exercise_price: 12.34, válido

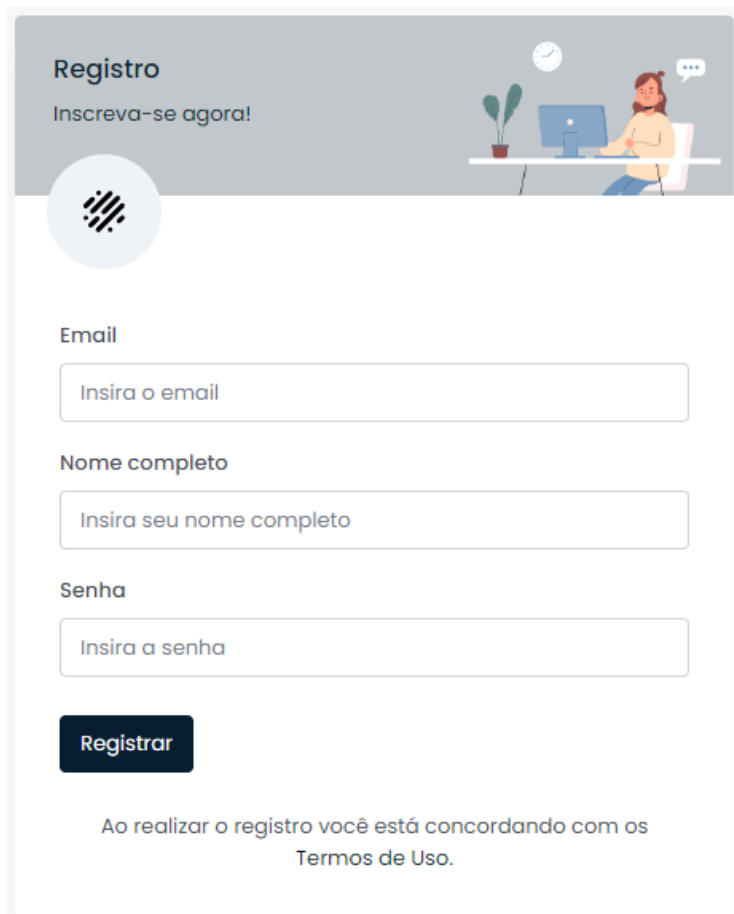
CT6: name: "12345678" , inválido
base: false , inválido
type: "9007199254740991" , inválido
date: null , inválido
date_expire: null , inválido
close_price: 10.11 , válido
exercise_price: 12.34, válido

CT7: name: "12345678" , inválido
base: false , inválido
type: "9007199254740991" , inválido
date: null , inválido
date_expire: null , inválido
close_price: -10.11 , inválido
exercise_price: 12.34, válido

CT8: name: "12345678" , inválido
base: false , inválido
type: "9007199254740991" , inválido
date: null , inválido
date_expire: null , inválido
close_price: -10.11 , inválido
exercise_price: true, inválido

7. TESTE DO CADASTRO DE USUÁRIO

Realiza a validação de interface de usuário - Cadastro de Usuário conforme a Figura seguir:



The screenshot shows a web form titled 'Registro' with the subtitle 'Inscreva-se agora!'. The form includes a header illustration of a person at a desk. Below the header is a circular logo with diagonal lines. The form fields are: 'Email' with a placeholder 'Insira o email', 'Nome completo' with a placeholder 'Insira seu nome completo', and 'Senha' with a placeholder 'Insira a senha'. A dark blue 'Registrar' button is positioned below the fields. At the bottom, a line of text states: 'Ao realizar o registro você está concordando com os Termos de Uso.'

Figura 8 - Tela de cadastro de usuário

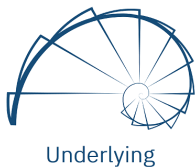
7.1.1 Restrições dos campos de cadastro de usuário

Requisitos de validação - Email:

- Possuir '@' no texto de inserção
- Possuir um ou mais '.' texto de inserção
- Possuir texto depois do '.'

Requisitos de validação - Nome completo:

- Não ser vazio
- Possuir mais de 3 caracteres



- Possuir no mínimo duas palavras (1º nome e sobrenome)

Requisitos de validação - Senha:

- Não ser vazia
- Possuir pelo menos 6 caracteres
- Possuir pelo menos 1 número
- Possuir pelo menos 1 caractere especial
- Possuir pelo menos 1 letra maiúscula
- Possuir pelo menos 1 letra minúscula

7.1.2 Teste Combinatorial do cadastro de usuário

Possíveis retornos de cada campo:

- **Email:** válido/inválido
- **Nome completo:** válido/inválido
- **Senha:** válido/inválido

Combinações de entradas: $2 \times 2 \times 2 = 8$

7.1.3 Suíte de Testes Combinatorial do cadastro de usuário

A seguir apresentado a realização dos testes combinatoriais para o cadastro de usuário:

Entradas válidas:

- **Email:** válido
- **Nome completo:** válido
- **Senha:** válido

CT1: 'leo@unifei.edu.br', 'Leo Sousa', '12345678'

Entradas inválidas:

- **Email:** inválido
- **Nome completo:** válido
- **Senha:** válido

CT2: 'leo@unifei', 'Leo Sousa', '#Abc123'

- **Email:** válido
- **Nome completo:** inválido
- **Senha:** válido

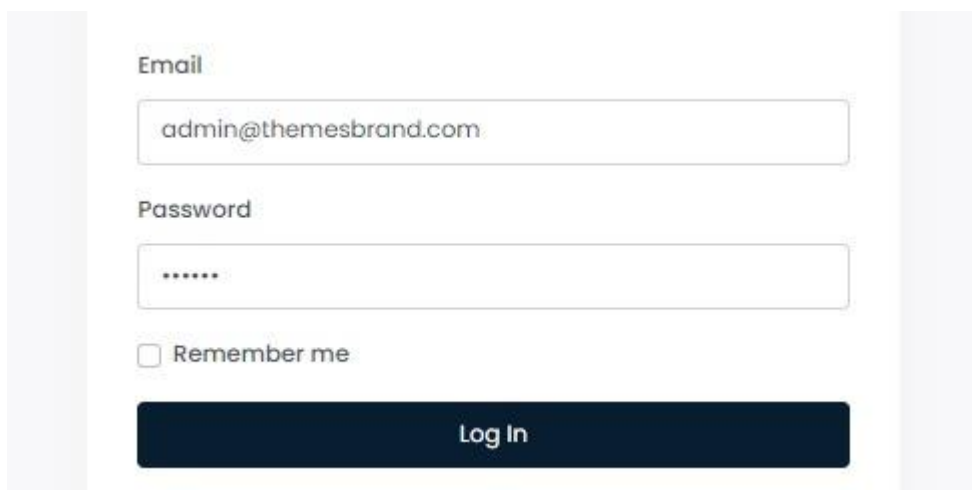
CT3: "leo@unifei.edu.br", 'Eu', '#Abc123'

- **Email:** válido
- **Nome completo:** válido
- **Senha:** inválido

CT4: "leo@unifei.edu.br", 'Leo Sousa', '#Abc'

8. TESTE DE AUTENTICAÇÃO DO USUÁRIO

Realiza a validação de interface de usuário - Autenticação do Usuário conforme a Figura seguir:



The image shows a user authentication form. It has two input fields: 'Email' and 'Password'. The 'Email' field contains the text 'admin@themesbrand.com'. The 'Password' field contains six asterisks. Below the password field is a checkbox labeled 'Remember me'. At the bottom of the form is a dark blue button labeled 'Log In'.

Figura 9 - Tela de autenticação de usuário

8.1.1 Restrições dos campos de autenticação do usuário

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 (.), sem espaço, sem acentuação e sem caracteres especiais
- No mínimo 3 e no máximo 10 caracteres depois do @; de A - Z ou de 0 - 9 e (_) e (.), sem espaço, sem acentuação e sem caracteres especiais

Para que uma senha seja considerada válida, deve:

- 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

No caso da senha e/ou o email ser inválido, a função deve retornar **false**.

8.1.2 Casos de testes de autenticação do usuário

Comprimento do email:

- o C1 – email com comprimento entre [3,20] caracteres antes do [@] e comprimento entre [3,20] após o [@] → válido
- o C2 email com comprimento fora dos limites [3,20] caracteres antes do [@] e comprimento entre [3,20] após o [@] → inválido
- o C3 – email com comprimento fora dos limites [3,20] caracteres antes do [@] e comprimento fora dos limites [3,20] após o [@] → inválido
- o C4 – email com comprimento entre [3,20] caracteres antes do [@] e comprimento fora dos limites [3,20] após o [@] → inválido
- o C5 – email vazio → inválido

Ocorrência de caracteres no email:

- C6 - comprimento entre [3,20] antes do [@], comprimento entre [3,20] após o [@] e com pelo menos uma ocorrência de pelo menos um caractere especial, sendo ele obrigatoriamente [@] e de letra [a-z,A-Z] → **válida**
- C7 - comprimento entre [3,20] antes do [@], comprimento entre [3,20] após o [@] e com pelo menos uma ocorrência de pelo menos um caractere especial, sendo ele obrigatoriamente [@] mas sem a ocorrência das letras [a-z,A-Z] → **inválida**

- C8 – Email sem a ocorrência do caractere obrigatório [@] → **inválida**
- C9 - Email sem a ocorrência do caractere obrigatório [@] e sem ocorrência das letras [a-z,A-Z] → **inválida**

Comprimento da senha:

- o C10 – senha de 8 caracteres → válido
- o C11 – senha de 20 caracteres → válido
- o C12 – senha com 7 caracteres → inválido
- o C13 – senha com 21 caracteres → inválido
- o C14 – senha vazio → inválido
- Ocorrências de caracteres na senha
 - o C15 - comprimento entre [8,20] e com pelo menos uma ocorrência de caractere especial [@, #, %, &, !,+], de número [0-9] e de letra [a-z,A-Z] → **válida**
 - o C16 - comprimento entre [8,20] e sem nenhuma ocorrência de caractere especial [@, #, %, &, !,+], nem de número [0-9] e nem de letra [a-z,A-Z] → **inválida**
 - o C17 - comprimento entre [8,20] e com alguma ocorrência de caractere especial [@, #, %, &, !,+], e de letra [a-z,A-Z] , mas sem ocorrência de número [0-9] → **inválida**
 - o C18 - comprimento entre [8,20] e com alguma ocorrência de letra [a-z,A-Z] e de número [0-9], mas sem ocorrência de caractere especial [@, #, %, &, !,+] → **inválida**

8.1.3 Suíte de Testes de autenticação do usuário

A Tabela a seguir apresentará a realização de cada teste planejado da secção anterior:

Classe de teste	Classes relacionadas	Caso de teste
CT1	C1	thiago2018002850@unifei.edu.br
CT2	C2	thiago20180028501321332@unifei.edu.br
CT3	C3	thiago20180028501321332@universidadefederaldeitajuba.edu.br
CT4	C4	thiago2018002850@ universidadefederaldeitajuba.edu.br
CT5	C5	vazio

CT6	C6	thiago2018002850@unifei.edu.br
CT7	C7	#34%*()@(32\$#!
CT8	C8	thiago2018002850unifei.edu.br
CT9	C9	#34%*() (32\$#!
CT10	C10	#34%*()1
CT11	C11	#34%*()@(32\$#!103412
CT12	C12	#34%*()
CT13	C13	#34%*()@(32\$#!1034120
CT14	C14	#34%*()1
CT15	C15	vazio
CT16	C16	thiagomp
CT17	C17	Thiagomp#
CT18	C18	Thiagomp10

Tabela 10 - Testes de autenticação do usuário

8.1.4 Teste Combinatorial de autenticação do usuário

Usando o critério de teste combinatorial, propor exemplos de casos de teste para o retorno da interface de autenticação de usuário.

- **Email:** válido/inválido
- **Senha:** válido/inválido

➤ **Combinações de entradas:** $2 \times 2 = 4$

8.1.5 Suíte de Testes Combinatorial de autenticação do usuário

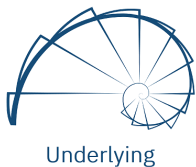
A seguir apresentado a realização dos testes combinatoriais para o retorno da interface de autenticação de usuário.

Entradas válidas:

- **Email:** válido
- **Senha:** válido

CT1: 'thiago2018002850@unifei.edu.br', '#34%*()1'

Entradas inválidas:



- **Email:** inválido
- **Senha:** válido

CT2: 'thiago2018002850@ universidadefederaldeitajuba.edu.br', '#34%*()1'

- **Email:** inválido
- **Senha:** inválido

CT3: 'thiago2018002850@ universidadefederaldeitajuba.edu.br', 'Thiagomp10'

- **Email:** válido
- **Senha:** inválido

CT3: 'thiago2018002850@unifei.edu.br', 'Thiagomp10'

9. TESTE DE INSERÇÃO DE OPÇÃO FICTÍCIA

Quando é desejado inserir uma opção fictícia na estratégia a ser criada, é necessário criar a opção. Para criar uma opção fictícia é necessário inserir um nome, escolher o tipo da operação e da transação, inserir o número de contratos, o preço da opção e o preço Underlying.

9.1.1 Restrições dos campos de inserção da opção fictícia:

Requisitos de validação - Nome:

- O nome deve ter mais de 3 caracteres

Requisitos de validação – N° de contratos:

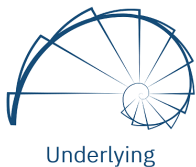
- Deve ser um número positivo
- Deve ser um número maior que zero
- Deve ser um número inteiro

Requisitos de validação – Preço da opção fictícia:

- Deve ser um número positivo
- Deve ser maior que 0,00 (podendo aceitar os decimais de zero)

Requisitos de validação – Preço Underlying da opção fictícia:

- Deve ser um número positivo
- Deve ser maior que 0,00 (podendo aceitar os decimais de zero)



Underlying



OBS: Os campos de tipo de operação e tipo de transação só possuem a restrição de não serem vazios, mas isso é impossível de acontecer. Eles são elementos de input HTML, do tipo <Select>, onde só existem duas opções e por padrão a opção “CALL” está inicialmente selecionada.

9.1.2 Teste Combinatorial de inserção da opção fictícia

Possíveis retornos de cada campo:

- **Nome:** válido/inválido
- **Tipo da operação:** válido
- **Tipo da transação:** válido
- **Número de contratos:** válido/inválido
- **Preço da opção:** válido/inválido
- **Preço Underlying:** válido/inválido

Combinação de entradas: $2*1*1*2*2*2 = 16$

9.1.3 Suíte de Testes Combinatorial de inserção da opção fictícia

A seguir apresentado a realização dos testes combinatórios para o cadastro de usuário:

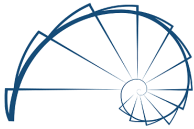
Entradas válidas:

- **Nome:** válido
- **Tipo da operação:** válido
- **Tipo da transação:** válido
- **Número de contratos:** válido
- **Preço da opção:** válido
- **Preço Underlying:** válido

CT1: “ABCD”, “CALL”, “LONG”, 3, 7.5, 2

Entradas inválidas:

- **Nome:** inválido
- **Tipo da operação:** válido
- **Tipo da transação:** válido



Underlying

- **Número de contratos:** válido
- **Preço da opção:** válido
- **Preço Underlying:** válido

CT2: “A”, “CALL”, “LONG”, 3, 7.5, 2

- **Nome:** válido
- **Tipo da operação:** válido
- **Tipo da transação:** válido
- **Número de contratos:** inválido
- **Preço da opção:** válido
- **Preço Underlying:** válido

CT3: “ABCD”, “CALL”, “LONG”, -3, 7.5, 2

Entradas inválidas:

- **Nome:** válido
- **Tipo da operação:** válido
- **Tipo da transação:** válido
- **Número de contratos:** válido
- **Preço da opção:** inválido
- **Preço Underlying:** válido

CT4: “ABCD”, “CALL”, “LONG”, 3, -7.5, 2

Entradas inválidas:

- **Nome:** válido
- **Tipo da operação:** válido
- **Tipo da transação:** válido
- **Número de contratos:** válido
- **Preço da opção:** válido
- **Preço Underlying:** inválido

CT5: “ABCD”, “CALL”, “LONG”, 3, 7.5, -2