Ícone

Descrição gerada automaticamente

**FACULDADE DE TECNOLOGIA DE COTIA**

**TECNOLOGIA EM DESENVOLVIMENTO DE SOFTWARE MULTIPLATAFORMA**

**THARSEO**

**PLANO DE TESTES**

**ALBERTO RIBEIRO DE OLIVEIRA, ALESSANDRA SANCHES, CAIO RIBEIRO, FLAVIO DOMINGUES VIEIRA, RICHARD GUEDES RIBEIRO DA SILVA**

**Cotia – SP**

**Junho/2025**

**FACULDADE DE TECNOLOGIA DE COTIA**

**THARSEO**

**PLANO DE TESTES**

**ALBERTO RIBEIRO DE OLIVEIRA, ALESSANDRA SANCHES, CAIO RIBEIRO, FLAVIO DOMINGUES VIEIRA, RICHARD GUEDES RIBEIRO DA SILVA**

**Professor (a) Orientador (a):Mario da Silva de Jesus**

**Cotia – SP**

**Junho/2025**

**Plano de Testes Tharseo Backend v2**

**Introdução**

O **Tharseo Backend v2** é uma API robusta, escalável e eficiente, projetada para suportar uma ampla variedade de recursos para a plataforma Tharseo. Esta API serve como o núcleo do sistema Tharseo, lidando com lógica de negócios, gerenciamento de banco de dados, autenticação e comunicação com clientes front-end e serviços externos. O Tharseo Backend v2 utiliza tecnologias modernas como **NestJS**, **Prisma ORM**, **Autenticação JWT** e **PostgreSQL** para fornecer um ambiente seguro, confiável e de alto desempenho para a plataforma.

O objetivo desse plano de testes, é elencar a cobertura dos métodos e fluxos de forma granular proporcionando segurança ao desenvolver e a prevenção de surpresas ao mudar determinados pontos do código que são usados por diversos métodos e repassar isso para produção.

**Métodos dos Testes**

Para realização dos testes a nível backend, utilizaremos o framework Jest e Supertest, que são bem conhecidos no mercado e apresentam um nível de confiança aceitável pela comunidade.

No momento, temos como alvo os controllers e suas rotas, serviços e seus métodos, classes e seus métodos.

Temos um total de testes unitários de 22 suítes com total de 162 testes. Também temos 4 suítes com 20 testes de integração (e2e).

Os testes unitários serão responsáveis em testar cada ponto da aplicação, em casos de sucesso e casos de falha.

Já os testes e2e, testarão o fluxo da integração, desde a rota recebida, regra de negócio validada, registro em banco de dados e retorno da resposta para o requisitante.

**Tipo de Testes**

* Testes Unitários - (Jest - NextJs)
* Testes de integração (e2e) - (SuperTest NextJs)

**Estratégia de Validação**

Para validação optamos em duas formas:  
 Para os testes unitários, ao desenvolver, podemos chamar os testes através de npm run test para testar localmente.

Para adicionar um duplo check, nossa pipeline tem como requisito antes de efetuar o deploy para produção, rodar todos os testes unitários e em caso de falha, o processo de deploy é abortado.

Para os testes de integração, os requisitos para rodar localmente, é instanciar um banco de dados postgres definindo as credenciais iguais as do arquivo env.test. Os testes serão iniciados e o feedback através do terminal.

Também adicionamos um duplo check em nossa pipeline, durante o deploy, o GitActions emula um banco de dados, em seguida, efetuamos um push de nossa estrutura do banco de dados para esse banco do GitActions, e ele é instanciado e permite nossa interação. Dessa forma, os testes e2e podem ser validados e em caso de falha o deploy é abortado.

**Cobertura dos Testes**

* 162 testes unitários
* 20 testes e2e
* **Execução e Descrição dos Testes Unitários**

**AuthController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 1 | auth.controller.spec.ts | Instancia | deve instanciar o auth.controller | sucesso | deve ser instanciado |
| 2 | auth.controller.spec.ts | signUp | deve criar e retornar um novo usuário | sucesso | deve retornar o usuário criado |
| 3 | auth.controller.spec.ts | signUp | deve retornar um erro se o usuario nao for criado | falha | deve retornar uma BadRequestException |
| 4 | auth.controller.spec.ts | signIn | deve retornar um payload de autenticação | sucesso | deve retornar um login com sucesso |
| 5 | auth.controller.spec.ts | signIn | deve retornar UnauthorizedException se o login falhar | falha | deve retornar UnauthorizedException |

**AuthGuard**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 6 | auth.guard.spec.ts | AuthGuard | deve instanciar o auth.guard | sucesso | deve ser instanciado |
| 7 | auth.guard.spec.ts | canActivate | deve retornar true se o token for válido | sucesso | deve retornar true |
| 8 | auth.guard.spec.ts | canActivate | deve lançar UnauthorizedException se o token não for encontrado | falha | deve retornar UnauthorizedException |
| 9 | auth.guard.spec.ts | canActivate | deve lançar UnauthorizedException se o token for inválido | falha | deve lançar UnauthorizedException |
| 10 | auth.guard.spec.ts | extractTokenFromHeader | deve extrair corretamente o token da header | sucesso | deve retornar o token |
| 11 | auth.guard.spec.ts | extractTokenFromHeader | deve retornar undefined se a header de authorization estiver ausente | falha | deve retornar o undefined |
| 12 | auth.guard.spec.ts | extractTokenFromHeader | deve retornar undefined se o tipo não for "Bearer | falha | deve retornar o undefined |

**AuthGuard**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 13 | auth.service.spec.ts | AuthService | deve instanciar o auth.service | sucesso | deve ser instanciado |
| 14 | auth.service.spec.ts | signUp | deve criar um usuário com sucesso | sucesso | deve retornar o usuário criado |
| 15 | auth.service.spec.ts | signUp | deve lançar erro ao falhar ao criar usuário | falha | deve retornar BadRequestException |
| 16 | auth.service.spec.ts | signIn | deve realizar login com sucesso | sucesso | deve retornar payload de sucesso no login |
| 17 | auth.service.spec.ts | signIn | deve lançar UnauthorizedException quando email não encontrado | falha | deve retornar UnauthorizedException |
| 18 | auth.service.spec.ts | signIn | 'eve lançar UnauthorizedException quando senha estiver incorreta | falha | deve retornar UnauthorizedException |

**BinanceApiController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 19 | binanceapi.controller.spec.ts | BinanceapiController | deve instanciar o binanceapi.controller | sucesso | deve ser instanciado |
| 20 | binanceapi.controller.spec.ts | check | deve retornar statusCode 200 e success | sucesso | deve retornar no body Ping Sucesso |

**BinanceApiService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 21 | binanceapi.service.spec.ts | BinanceapiService | deve instanciar o binanceapi.service | sucesso | deve ser instanciado |
| 22 | binanceapi.service.spec.ts | getAllOrders | deve retornar statusCode 200 e successdeve buscar todas as ordens com sucesso | sucesso | deve buscar ordens para BTCUSDT |
| 23 | binanceapi.service.spec.ts | getAllOrders | deve lançar exceção se der erro ao buscar ordens | falha | deve retornar InternalServerErrorException |
| 24 | binanceapi.service.spec.ts | newOrder | deve executar uma ordem MARKET com sucesso | sucesso | deve retornar response a ordem mocada |
| 25 | binanceapi.service.spec.ts | newOrder | deve executar uma ordem LIMIT com sucesso | sucesso | deve retornar response a ordem mocada |
| 26 | binanceapi.service.spec.ts | newOrder | deve lançar BadRequestException se tipo de ordem for inválido | falha | deve retornar InternalServerErrorException |
| 27 | binanceapi.service.spec.ts | newOrder | deve lançar InternalServerErrorException se houver erro na execução | falha | deve retornar InternalServerErrorException |
| 28 | binanceapi.service.spec.ts | cancelOrder | deve cancelar uma ordem com sucesso | sucesso | deve retornar response a ordem mocada |
| 29 | binanceapi.service.spec.ts | cancelOrder | deve lançar exceção ao falhar no cancelamento da ordem | falha | deve retornar no response: Erro ao cancelar ordem |
| 30 | binanceapi.service.spec.ts | cancelOpenOrders | deve cancelar todas as ordens abertas com sucesso | sucesso | deve retornar no response a ordem cancelada |
| 31 | binanceapi.service.spec.ts | cancelOpenOrders | deve lançar exceção ao falhar no cancelamento de ordens abertas | falha | deve retornar no response: Erro ao cancelar ordem |
| 32 | binanceapi.service.spec.ts | getPriceMarket | deve retornar o preço de mercado com sucesso | sucesso | deve retornar o preco do ativo (28000.50) |
| 33 | binanceapi.service.spec.ts | getPriceMarket | deve lançar exceção ao falhar ao buscar preço de mercado | falha | deve retornar Erro ao obter preço de mercado para BTCUSDT |
| 34 | binanceapi.service.spec.ts | checkOrder | deve verificar o status da ordem com sucesso | sucesso | deve retornar o mock de resposta da ordem |
| 35 | binanceapi.service.spec.ts | checkOrder | deve lançar exceção ao falhar ao verificar a ordem | falha | deve retornar Erro ao verificar a ordem... |

**PrismaService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 36 | prisma.service.spec.ts | PrismaService | deve instanciar o prisma.service | sucesso | deve ser instanciado |

**Engine-TharseoController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 37 | engine-tharseo.controller.spec.ts | EngineTharseoController | deve instanciar o engine-tharseo.controller | sucesso | deve ser instanciado |
| 38 | engine-tharseo.controller.spec.ts | startEngine | deve chamar startEngineTharseo no controller | sucesso | deve retornar statuscode 200 na rota |
| 39 | engine-tharseo.controller.spec.ts | checkOrders | deve chamar checkOrders no controller | sucesso | deve retornar statuscode 200 na rota |

**Engine-TharseoService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 40 | engine-tharseo.service.spec.ts | EngineTharseoService | deve instanciar o engine-tharseo.service | sucesso | deve ser instanciado |
| 41 | engine-tharseo.service.spec.ts | getOpenOrders | deve retornar as ordens de venda pendentes | sucesso | retornar ordens do mock |
| 42 | engine-tharseo.service.spec.ts | getActivesTradeflow | deve retornar os tradeflows ativos | sucesso | retonar tradeflows do mock |
| 43 | engine-tharseo.service.spec.ts | sendExchangeOrder | deve enviar a ordem para a Exchange | sucesso | retornar ordem enviada do mock |
| 44 | engine-tharseo.service.spec.ts | createOrder | deve criar uma nova ordem | sucesso | retornar ordem criado do mock |

**HealthController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 45 | health.controller.spec.ts | HealthController | deve instanciar o health.controller | sucesso | deve ser instanciado |
| 46 | health.controller.spec.ts | check | deve retornar que o servidor está operacional | sucesso | deve retornar: Application is up and running |
| 47 | health.controller.spec.ts | check | deve retornar um InternalServerErrorException indicando que não está operacional | falha | deve retornar InternalServerErrorException |

**OrderController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 48 | order.controller.spec.ts | OrderController | deve instanciar o order.controller | sucesso | deve ser instanciado |
| 49 | order.controller.spec.ts | create | deve criar um pedido | sucesso | deve retornar o pedido criado |
| 50 | order.controller.spec.ts | create | deve lançar um NotFoundException se o usuario nao for encontrado | falha | deve retornar NotFoundException |
| 51 | order.controller.spec.ts | findAll | deve retornar uma lista de orders | sucesso | deve retornar as ordens do mock |
| 52 | order.controller.spec.ts | findOne | deve retornar um order | sucesso | deve retornar a ordem mocada |
| 53 | order.controller.spec.ts | findOne | deve retornar um NotFoundException se o order nao for encontrado | falha | deve retornar NotFoundException |
| 54 | order.controller.spec.ts | findOne | deve retornar um NotFoundException se o usuario nao for encontrado | falha | deve retornar NotFoundException |
| 55 | order.controller.spec.ts | update | deve atualizar uma ordem | sucesso | deve retornar a ordem atualizada do mock |
| 56 | order.controller.spec.ts | update | deve retornar um NotFoundException se o order nao for encontrado | falha | deve retornar NotFoundException |
| 57 | order.controller.spec.ts | update | deve retornar um NotFoundException se o usuario nao for encontrado | falha | deve retornar NotFoundException |
| 58 | order.controller.spec.ts | cancelOpenOrders | deve cancelar todas as ordens abertas | sucesso | deve retornar {success: true,  message: 'Ordens canceladas com sucesso',} |
| 59 | order.controller.spec.ts | cancelOpenOrders | deve retornar um NotFoundException se o usuario nao for encontrado' | falha | deve retornar NotFoundException |

**OrderService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 60 | order.service.spec.ts | OrderService | deve instanciar o order.service | sucesso | deve ser instanciado |
| 61 | order.service.spec.ts | create | deve criar uma order com sucesso | sucesso | deve retornar a ordem do mock criada |
| 62 | order.service.spec.ts | create | deve lançar NotFoundException se asset não for encontrado | falha | deve retornar NotFoundException |
| 63 | order.service.spec.ts | create | deve lançar NotFoundException se user não for encontrado | falha | deve retornar NotFoundException |
| 64 | order.service.spec.ts | create | deve lançar ServiceUnavailableException se status da ordem não for FILLED | falha | deve retornar ServiceUnavailableException |
| 65 | order.service.spec.ts | create | deve lançar erro se prisma.order.create falhar | falha | deve retornar Erro ao salvar |
| 66 | order.service.spec.ts | createOnDatabase | deve criar uma order com sucesso | sucesso | deve retornar a ordem do mock criada |
| 67 | order.service.spec.ts | createOnDatabase | deve lançar NotFoundException se asset nao for encontrado | falha | deve retornar NotFoundException |
| 68 | order.service.spec.ts | createOnDatabase | deve lançar NotFoundException se user nao for encontrado | falha | deve retornar NotFoundException |
| 69 | order.service.spec.ts | findAll | deve retornar todas as orders de um usuario | sucesso | deve retornar as ordens do mock |
| 70 | order.service.spec.ts | findAll | deve retornar um array vazio se o usuario nao tiver orders | sucesso | deve retornar um array vazio |
| 71 | order.service.spec.ts | findAll | deve retornar NotFoundException se a order nao for encontrada | falha | deve retornar NotFoundException |
| 72 | order.service.spec.ts | update | deve atualizar uma order | sucesso | deve retornar a ordem atualizada do mock |
| 73 | order.service.spec.ts | update | deve retornar NotFoundException se a order nao for encontrada | falha | deve retornar NotFoundException |
| 74 | order.service.spec.ts | updateIdPairOrder | deve atualizar uma order com atribuição de pairOrderId | sucesso | deve retornar a ordem atualizada do mock |
| 75 | order.service.spec.ts | getPendingOrdersCreated | deve retornar uma lista de orders pendentes | sucesso | deve retornar as ordens do mock |
| 76 | order.service.spec.ts | cancelOpenOrders | deve cancelar ordens abertas do usuário e atualizar o status para CANCELADO | sucesso | deve retornar as ordens atualizadas com status CANCELADO do mock |
| 77 | order.service.spec.ts | updateOrderFromCheckExchange | deve atualizar a ordem e a estratégia se order tiver result, strategyId e performance | sucesso | deve retornar a ordem atualizada do mock |

**StrategyController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 78 | strategy.controller.spec.ts | StrategyController | deve instanciar o strategy.controller | sucesso | deve ser instanciado |
| 79 | strategy.controller.spec.ts | create | deve criar uma strategy | sucesso | deve retornar a strategy mocada |
| 80 | strategy.controller.spec.ts | create | deve retornar um erro se o usuario nao for encontrado | falha | deve retornar NotFoundException |
| 81 | strategy.controller.spec.ts | findAll | deve retornar todas as strategies | sucesso | deve retornar as strategys do mock |
| 82 | strategy.controller.spec.ts | findAll | deve retornar um erro se o usuario nao for encontrado | falha | deve retornar NotFoundException |
| 83 | strategy.controller.spec.ts | findOne | deve retornar uma strategy | sucesso | deve retornar a strategy do mock |
| 84 | strategy.controller.spec.ts | findOne | deve retornar um NotFoundException se a strategy nao for encontrada | falha | deve retornar NotFoundException |
| 85 | strategy.controller.spec.ts | findOne | deve retornar um erro se o usuario nao for encontrado | falha | deve retornar NotFoundException |
| 86 | strategy.controller.spec.ts | update | deve atualizar uma strategy | sucesso | deve retornar a strategy atualizada do mock |
| 87 | strategy.controller.spec.ts | update | deve retornar um NotFoundException se a strategy nao for encontrada | falha | deve retornar NotFoundException |
| 88 | strategy.controller.spec.ts | update | deve retornar um erro se o usuario nao for encontrado | falha | deve retornar NotFoundException |

**StrategyService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 89 | strategy.service.spec.ts | StrategyService | deve instanciar o strategy.service | sucesso | deve ser instanciado |
| 90 | strategy.service.spec.ts | create | deve criar uma nova strategy com sucesso | sucesso | deve retornar a strategy mocada |
| 91 | strategy.service.spec.ts | create | deve lançar um erro se o usuário não for encontrado | falha | deve retornar NotFoundException |
| 92 | strategy.service.spec.ts | findAll | deve retornar todas as strategies do usuario que requisitou | sucesso | deve retornar as strategies mocadas |
| 93 | strategy.service.spec.ts | findAll | deve retornar NotFoundException se o usuário nao for encontrado | falha | deve retornar NotFoundException |
| 94 | strategy.service.spec.ts | findOne | deve retornar uma strategy pelo id e pelo usuário | sucesso | deve retornar a strategy do |
| 95 | strategy.service.spec.ts | findOne | deve retornar NotFoundException se o usuario nao for encontrado | falha | deve retornar NotFoundException |
| 96 | strategy.service.spec.ts | findOne | deve retornar NotFoundException se a strategy nao for encontrada | falha | deve retornar NotFoundException |
| 97 | strategy.service.spec.ts | update | deve atualizar uma strategy pelo id e pelo usuário | sucesso | deve retornar a strategy atualizada do mock |
| 98 | strategy.service.spec.ts | update | deve retornar NotFoundException se o usuario nao for encontrado | falha | deve retornar NotFoundException |
| 99 | strategy.service.spec.ts | update | deve retornar NotFoundException se a strategy nao for encontrada | falha | deve retornar NotFoundException |

**AssetController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 100 | asset.controller.spec.ts | AssetController | deve instanciar o asset.controller | sucesso | deve ser instanciado |
| 101 | asset.controller.spec.ts | create | deve criar um novo ativo | sucesso | valor retornado deve ser igual ao esperado |
| 102 | asset.controller.spec.ts | findAll | deve retornar todos os ativos | sucesso | retornar lista de ativos mocados |
| 103 | asset.controller.spec.ts | findOne | deve retornar o ativo pelo ID | sucesso | retornar asset igual ao mocado |
| 104 | asset.controller.spec.ts | findOne | deve lançar exceção se o ativo não existir | falha | retornar NotFoundException |
| 105 | asset.controller.spec.ts | update | deve atualizar o ativo se ele existir | sucesso | retornar asset atualizado conforme mock |
| 106 | asset.controller.spec.ts | update | deve lançar exceção se o ativo não existir | falha | retornar NotFoundException |
| 107 | asset.controller.spec.ts | remove | deve remover o ativo se ele existir | sucesso | retornar asset igual ao mocado |
| 108 | asset.controller.spec.ts | remove | deve lançar exceção se o ativo não existir | falha | retornar NotFoundException |

**AssetService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 109 | asset.service.spec.ts | Instancia | deve instanciar o asset.service | sucesso | deve ser instanciado |
| 110 | asset.service.spec.ts | create | deve criar e retornar um novo ativo | sucesso | deve criar um asset |
| 111 | asset.service.spec.ts | findAll | deve retornar uma lista de ativos | sucesso | deve retornar uma lista de assets |
| 112 | asset.service.spec.ts | findOne | deve retornar um ativo pelo id | sucesso | deve retornar o asset |
| 13 | asset.service.spec.ts | findOne | deve retornar null se o ativo não for encontrado | falha | deve retornar null |
| 114 | asset.service.spec.ts | update | deve atualizar e retornar o ativo | sucesso | deve retornar o asset atualizado |
| 115 | asset.service.spec.ts | remove | deve remover e retornar o ativo removido | sucesso | deve retornar o asset removido |

**TradeflowController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 116 | tradeflow.controller.spec.ts | TradeflowController | deve instanciar o tradeflow.controller | sucesso | deve ser instanciado |
| 117 | tradeflow.controller.spec.ts | create | deve criar um novo TradeFlow com sucesso | sucesso | deve retornar o tradeflow do mock |
| 118 | tradeflow.controller.spec.ts | findAll | deve retornar uma lista de tradeflows | sucesso | deve retornar uma lista de tradeflows |
| 119 | tradeflow.controller.spec.ts | findOne | deve retornar um tradeflow | sucesso | deve retornar o tradeflow do mock |
| 120 | tradeflow.controller.spec.ts | findOne | deve retornar uma exception se o tradeflow nao for encontrado | falha | deve retornar NotFoundException |
| 121 | tradeflow.controller.spec.ts | update | deve atualizar um tradeflow | sucesso | deve retornar o tradeflow atualizado do mock |
| 122 | tradeflow.controller.spec.ts | update | deve retornar uma exception se o tradeflow nao for encontrado | falha | deve retornar NotFoundException |
| 123 | tradeflow.controller.spec.ts | remove | deve remover um tradeflow com sucesso | sucesso | deve retornar o tradeflow excluido |
| 124 | tradeflow.controller.spec.ts | remove | deve lançar NotFoundException se o tradeflow não for encontrado | falha | deve retornar NotFoundException |
| 125 | tradeflow.controller.spec.ts | remove | deve lançar UnauthorizedException se o usuário não for o dono do tradeflow | sucesso | deve retornar UnauthorizedException |

**TradeflowService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 126 | tradeflow.service.spec.ts | TradeflowService | deve instanciar o tradeflow.service | sucesso | deve ser instanciado |
| 127 | tradeflow.service.spec.ts | create | deve criar um novo tradeflow com sucesso | sucesso | deve retornar o tradeflow criado do mock |
| 128 | tradeflow.service.spec.ts | create | deve lançar NotFoundException se a estratégia não for encontrada | falha | deve retornar NotFoundException |
| 129 | tradeflow.service.spec.ts | create | deve lançar NotFoundException se o ativo não for encontrado | falha | deve retornar NotFoundException |
| 130 | tradeflow.service.spec.ts | findAll | deve retornar todos os tradeflows do usuário | sucesso | deve retornar os tradeflows do mock |
| 131 | tradeflow.service.spec.ts | findOne | deve retornar o tradeflow se encontrado | sucesso | deve retornar o tradeflow do mock |
| 132 | tradeflow.service.spec.ts | findOne | deve lançar NotFoundException se o tradeflow não for encontrado | falha | deve retornar NotFoundException |
| 133 | tradeflow.service.spec.ts | update | deve atualizar o tradeflow se existir e pertencer ao usuário | sucesso | deve retornar o tradeflow atualizado do mock |
| 134 | tradeflow.service.spec.ts | update | deve lançar NotFoundException se o tradeflow não existir | falha | deve retornar NotFoundException |
| 135 | tradeflow.service.spec.ts | update | deve lançar UnauthorizedException se o tradeflow não pertencer ao usuário | falha | deve retornar UnauthorizedException |
| 136 | tradeflow.service.spec.ts | remove | deve remover o tradeflow se existir e pertencer ao usuário | sucesso | deve retornar o tradeflow removido do mock |
| 137 | tradeflow.service.spec.ts | remove | deve lançar NotFoundException se o tradeflow não existir | falha | deve retornar NotFoundException |
| 138 | tradeflow.service.spec.ts | remove | deve lançar UnauthorizedException se o tradeflow não pertencer ao usuário | falha | deve retornar UnauthorizedException |

**UserController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 139 | user.controller.spec.ts | UserController | deve instanciar o user.controller | sucesso | deve ser instanciado |
| 140 | user.controller.spec.ts | getAllUsers | deve retornar dois usuários | sucesso | deve retornar dois usuarios do mock |

**UserService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 141 | user.service.spec.ts | UserService | deve instanciar o user.service | sucesso | deve ser instanciado |
| 142 | .service.spec.ts |  | deve criar um novo usuário e retornar os dados sem a senha | sucesso | deve retornar o usuário criado |
| 143 | user.service.spec.ts | getUserByEmail | deve retornar um usuário se encontrado por email | sucesso | deve retornar o usuário do mock |
| 144 | user.service.spec.ts | getUserByEmail | deve retornar null se o usuário não for encontrado | falha | deve retornar null |
| 145 | user.service.spec.ts | getAllUsers | deve retornar uma lista de usuários | sucesso | deve retornar um array de usuários do mock |
| 146 | user.service.spec.ts | getUserById | deve retornar um usuário com wallets e credential se encontrado por ID | sucesso | deve retornar o usuário do mock |
| 147 | user.service.spec.ts | getUserById | deve retornar null se o usuário não for encontrado | falha | deve retornar null |

**WalletController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 148 | wallet.controller.spec.ts | WalletController | deve instanciar o wallet.controller | sucesso | deve ser instanciado |
| 149 | wallet.controller.spec.ts | createOrUpdate | deve chamar walletService.createOrUpdate com os parâmetros corretos | sucesso | deve retornar a wallet criada do mock |
| 150 | wallet.controller.spec.ts | findAll | deve retornar uma lista de carteiras do usuário | sucesso | deve retornar uma lista de wallets do mock |
| 151 | wallet.controller.spec.ts | findOne | deve retornar uma carteira específica do usuário pelo ID | sucesso | deve retornar a wallet mocada |
| 152 | wallet.controller.spec.ts | update | deve chamar walletService.update com os parâmetros corretos | sucesso | deve retornar a wallet atualizada do mock |
| 153 | wallet.controller.spec.ts | remove | deve chamar walletService.remove com o ID correto | sucesso | deve retornar a wallet removida mocada |

**WalletService**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Método/Função** | **Objetivo** | **Caso** | **Resultado** |
| 154 | wallet.service.spec.ts | WalletService | deve instanciar o wallet.service | sucesso | deve ser instanciado |
| 155 | wallet.service.spec.ts | createOrUpdate | deve criar uma nova wallet se não existir | sucesso | deve retornar uma wallet criada pelo mock |
| 156 | wallet.service.spec.ts | createOrUpdate | deve atualizar uma carteira existente | sucesso | deve atualizar uma wallet existente com o mock |
| 157 | wallet.service.spec.ts | createOrUpdate | deve lançar uma excessão se um usuário não for encontrado | falha | deve retornar NotFoundException |
| 158 | wallet.service.spec.ts | createOrUpdate | deve lançar uma excessão se um ativo nao for encontrado | falha | deve retornar NotFoundException |
| 159 | wallet.service.spec.ts | updateWallet | deve atualizar e retornar uma wallet | sucesso | deve retornar uma wallet atualizada com o mock |
| 160 | wallet.service.spec.ts | createWallet | deve criar e retornar uma nova wallet | sucesso | deve criar uma wallet mocada |
| 161 | wallet.service.spec.ts | findAll | deve retornar todas as carteiras do usuário | sucesso | deve retornar uma lista de wallets mocada |
| 162 | wallet.service.spec.ts | findOne | deve retornar se existir, uma carteira do usuário | sucesso | deve retornar a wallet mocada de um usuário |
| 163 | wallet.service.spec.ts | findOne | deve lançar uma excessão se a wallet não pertencer ao usuário | falha | deve retornar NotFoundException |

* **Execução e Descrição dos Testes de Integração (E2E)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Núm.** | **Classe** | **Objetivo** | **Fluxos** | **Caso** | **Resultado** |
| 1 | App.e2e-spec.ts | Verificar se a aplicação inicia e fica disponível para sem erros | GET /health/check  → Aciona: check – HealthController | sucesso | deve receber na rota status code 200 e a mensagem: Application is up and running |
| 2 | Auth.e2e-spec.ts | Registrar um novo usuário | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – userService  → Aciona: create – prismaService  → Registra: User – PostgreSQL | sucesso | deve registrar um usuário e receber de volta o registro com o payload processado |
| 3 | Auth.e2e-spec.ts | Registrar um novo usuário com e-mail já existente | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – userService | falha | deve receber status code 400 com a mensagem Erro ao cadastrar usuário |
| 4 | Auth.e2e-spec.ts | Fazer Login na Aplicação com Credenciais Corretas | POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – userService  → Aciona: findFirst – prismaService  → Aciona: compare – bcrypt | sucesso | deve receber um payload com dados principais, token e expiração do token |
| 5 | Auth.e2e-spec.ts | Fazer Login na Aplicação com Credenciais Incorretas | POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – userService  → Aciona: findFirst – prismaService  → Busca: User – PostgreSQL  → Aciona: compare – bcrypt | falha | deve receber Unauthorized com um JSON com campos: message, error e statusCode |
| 6 | Strategy.e2e-spec.ts | Criar uma nova estratégia com Sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/strategy  → Aciona: create – StrategyController  → Aciona: create – StrategyService  → Aciona: getUserById – UserService  → Aciona: create – PrismaService  → Cria: Strategy – PostgreSQL | sucesso | deve receber um payload com detalhes da estratégia criada |
| 7 | Strategy.e2e-spec.ts | Criar uma nova estratégia sem sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/strategy  → Aciona: create – StrategyController | falha | deve receber um status code 400 com payload com propriedades: message, error e statusCode com a message = ‘Bad Request’ |
| 8 | Strategy.e2e-spec.ts | Consultar todas estratégias do usuário com sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/strategy  → Aciona: create – StrategyController  → Aciona: create – StrategyService  → Aciona: getUserById – UserService  → Aciona: create – PrismaService  → Cria: Strategy – PostgreSQL  GET /v1/strategy  → Aciona: findAll – StrategyController  → Aciona: findAll – StrategyService  → Aciona: getUserById – UserService  → Aciona: findMany – PrismaService  → Busca: Strategy – PostgreSQL | sucesso | deve receber a lista de estratégias do usuário ou uma lista vazia |
| 9 | Strategy.e2e-spec.ts | Consultar uma estratégia do usuário pelo Id com sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/strategy  → Aciona: create – StrategyController  → Aciona: create – StrategyService  → Aciona: getUserById – UserService  → Aciona: create – PrismaService  → Cria: Strategy – PostgreSQL  GET /v1/strategy/{id}  → Aciona: findOne – StrategyController  → Aciona: findOne – StrategyService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Strategy – PostgreSQL | sucesso | deve receber a estratégia que foi consultada pelo id |
| 10 | Strategy.e2e-spec.ts | Deve atualizar uma estratégia pelo id e body com sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/strategy  → Aciona: create – StrategyController  → Aciona: create – StrategyService  → Aciona: getUserById – UserService  → Aciona: create – PrismaService  → Cria: Strategy – PostgreSQL  PATCH /v1/strategy/{id}  → Aciona: update – StrategyController  → Aciona: update – StrategyService  → Aciona: findUnique – PrismaService  → Aciona: update – PrismaService  → Busca: Strategy – PostgreSQL | sucesso | deve receber a estratégia que foi consultada pelo id |
| 11 | Tradeflow.e2e-spec.ts | Deve criar uma nova automação com sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/tradeflow  → Aciona: create – TradeflowController  → Aciona: create – TradeflowService  → Aciona: findOne – StrategyService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Strategy – PostgreSQL  → Aciona: findOne – AssetService  → Aciona: findUnique – PrismaService  → Busca: Asset – PostgreSQL  → Aciona: create – PrismaService  → Cria: Tradeflow – PostgreSQL | sucesso | deve receber a automação criada no payload da requisição |
| 12 | Tradeflow.e2e-spec.ts | Deve criar uma nova automação sem sucesso – faltando o token | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/tradeflow  → Aciona: create – TradeflowController | falha | deve receber UnauthorizedException como resposta contendo no payload message, error e statusCode |
| 13 | Tradeflow.e2e-spec.ts | Deve criar uma nova automação sem sucesso – Asset não encontrado | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/tradeflow  → Aciona: create – TradeflowController  → Aciona: create – TradeflowService  → Aciona: findOne – StrategyService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Strategy – PostgreSQL  → Aciona: findOne – AssetService  → Aciona: findUnique – PrismaService  → Busca: Asset – PostgreSQL | falha | deve receber NotFoundException como resposta contendo no payload message, error e statusCode |
| 14 | Tradeflow.e2e-spec.ts | Deve criar uma nova automação sem sucesso – Strategy não encontrado | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/tradeflow  → Aciona: create – TradeflowController  → Aciona: create – TradeflowService  → Aciona: findOne – StrategyService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Strategy – PostgreSQL | falha | deve receber NotFoundException como resposta contendo no payload message, error e statusCode |
| 15 | Tradeflow.e2e-spec.ts | Deve criar uma nova automação sem sucesso – Strategy não encontrado | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/tradeflow  → Aciona: create – TradeflowController  → Aciona: create – TradeflowService  → Aciona: findOne – StrategyService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Strategy – PostgreSQL | falha | deve receber NotFoundException como resposta contendo no payload message, error e statusCode |
| 16 | Tradeflow.e2e-spec.ts | Deve retornar uma lista de Tradeflows de um usuário com sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/tradeflow  → Aciona: create – TradeflowController  → Aciona: create – TradeflowService  → Aciona: findOne – StrategyService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Strategy – PostgreSQL  → Aciona: findOne – AssetService  → Aciona: findUnique – PrismaService  → Busca: Asset – PostgreSQL  → Aciona: create – PrismaService  → Cria: Tradeflow – PostgreSQL  GET /v1/tradeflow  → Aciona: findAll – TradeflowController  → Aciona: findAll – TradeflowService  → Aciona: getUserById – UserService  → Aciona: findMany – PrismaService  → Busca: Tradeflow – PostgreSQL | sucesso | deve receber uma lista de tradeflows |
| 17 | Tradeflow.e2e-spec.ts | Deve retornar uma lista vazia para um usuário sem tradeflows com sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  GET /v1/tradeflow  → Aciona: findAll – TradeflowController  → Aciona: findAll – TradeflowService  → Aciona: getUserById – UserService  → Aciona: findMany – PrismaService  → Busca: Tradeflow – PostgreSQL | sucesso | deve receber uma lista vazia |
| 18 | Tradeflow.e2e-spec.ts | Deve retornar um tradeflow de um usuário pesquisado pelo id com sucesso | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  POST /v1/tradeflow  → Aciona: create – TradeflowController  → Aciona: create – TradeflowService  → Aciona: findOne – StrategyService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Strategy – PostgreSQL  → Aciona: findOne – AssetService  → Aciona: findUnique – PrismaService  → Busca: Asset – PostgreSQL  → Aciona: create – PrismaService  → Cria: Tradeflow – PostgreSQL  GET /v1/tradeflow/{id}  → Aciona: findOne – TradeflowController  → Aciona: findOne – TradeflowService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Tradeflow – PostgreSQL | sucesso | deve receber o tradeflow referente ao id |
| 19 | Tradeflow.e2e-spec.ts | Deve retornar NotFoundException para um tradeflow não existente | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  GET /v1/tradeflow/{id}  → Aciona: findOne – TradeflowController  → Aciona: findOne – TradeflowService  → Aciona: getUserById – UserService  → Aciona: findUnique – PrismaService  → Busca: Tradeflow – PostgreSQL | falha | deve receber NotFoundException para o id inexistente |
| 20 | Tradeflow.e2e-spec.ts | Deve retornar UnauthorizedException para um tradeflow não existente | POST /auth/signup  → Aciona: signUp – AuthController  → Aciona: signUp – AuthService  → Aciona: createUser – UserService  → Aciona: create – PrismaService  → Registra: User – PostgreSQL  POST /auth/signin  → Aciona: signIn – AuthController  → Aciona: getUserByEmail – UserService  → Aciona: findFirst – PrismaService  → Busca: User – PostgreSQL  → Aciona: compare – Bcrypt  GET /v1/tradeflow/{id}  → Aciona: findOne – TradeflowController | falha | deve receber UnauthorizedException para o id inexistente |

**Plano de Testes Tharseo Frontend v2**

**Introdução**

Utilizamos da tecnologia Cypress para realizar o teste do frontend nos principais fluxos utilizados pelos usuários como cadastro, login, navegação por diversas partes funcionais da aplicação etc. a partir do algoritmo o Cypress automatizou clicando e preenchendo dados nos campos e botões definidos no algoritmo, gerando automação e facilidade nos testes, de modo que ao ser modificada a aplicação gera somente a necessidade de nova execução do algoritmo sem a necessidade de teste manual do QA, garantindo que as modificações feitas em atualizações não irão impactar no que já estava funcional.

**Métodos dos Testes**

O Cypress é uma ferramenta moderna de testes end-to-end (E2E) voltada para aplicações web. Ele permite simular interações reais de usuários com a interface da aplicação, garantindo que os fluxos funcionem corretamente. Diferente de outras ferramentas, o Cypress roda diretamente no navegador, junto com a aplicação, o que proporciona acesso completo ao DOM, console e rede, permitindo testes mais precisos e rápidos.

Os testes são escritos em JavaScript ou TypeScript e seguem uma estrutura clara e encadeada. Um teste básico geralmente envolve visitar uma URL, interagir com elementos da página (como campos de texto e botões) e realizar asserções para verificar se o comportamento esperado foi alcançado. Por exemplo, um teste de login pode incluir comandos como cy.visit() para acessar a página, cy.get() para selecionar elementos, cy.type() para preencher campos e cy.click() para acionar botões. As verificações são feitas com cy.should(), que valida o estado da aplicação após as interações.

Uma das principais vantagens do Cypress é sua capacidade de esperar automaticamente por elementos estarem disponíveis antes de interagir com eles, eliminando a necessidade de comandos manuais de espera. Além disso, a ferramenta oferece uma interface visual que mostra cada etapa do teste em tempo real, facilitando a depuração e o entendimento do fluxo de execução.

Em resumo, o método de testes do Cypress é baseado em comandos encadeados, observação do DOM e espera inteligente, proporcionando uma abordagem confiável, legível e eficiente para garantir a qualidade de aplicações web.

**Estratégia de Validação**

Optamos pelo teste dos principais fluxos e funcionalidades da nossa aplicação garantindo a funcionalidade nos processos de cadastro, login, e das principais funcionalidades que já estão funcionais na nossa aplicação, segue abaixo o plano de testes realizados através do Cypress:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cenário** | **Caso de teste** | **BDD** | **Status** |
| 001-Cadastro | CT001.001 - Cadastro com e-mail inválido | Dado que eu esteja na Tela de cadastro Quando preencher o campo e-mail com um e-mail invalido sem @ Então o sistema deve apontar o erro de e-mail inválido | Passou |
| CT001.002 - Cadastro com e-mail válido | Dado que eu esteja na Tela de cadastro Quando preencher o campo e-mail com um e-mail valido sem @ Então o sistema deve deixar eu dar continuidade no processo de cadastro | Passou |
| CT001.003- Cadastro um senha que não atende os requisitos de complexidade | Dado que eu esteja na Tela de cadastro Quando preencher o campo de senha com determinada senha que não atenda os requisitos determinados  Então o sistema deve apontar o erro de senha inválida | Passou |
| CT001.004 - Cadastro um senha que atende os requisitos de complexidade | Dado que eu esteja na Tela de cadastro Quando preencher o campo de senha com determinada senha atenda os requisitos determinados  Então o sistema deve permitir que prossigamos com o processo de cadastro | Passou |
| CT001.005 - Cadastro de um número celular invalido | Dado que eu esteja na Tela de cadastro Quando preencher o campo de numero de celular invalido Então o sistema deve acusar número invalido | Passou |
| CT001.006 - Cadastro de um número celular valido | Dado que eu esteja na Tela de cadastro Quando preencher o campo de numero de celular invalido Então o sistema deve permitir que sigamos com o processo de cadastro | Passou |
| CT001.007 - Teste de cadastro com um e-mail que já foi cadastrado | Dado que eu esteja na Tela de cadastro Quando for finalizar o cadastro utilizando um e-mail já cadastrado Então o sistema deve apontar erro do processo de cadastro | Passou |
| 002-Login | CT002.001 - login com e-mail inválido | Dado que eu esteja na tela de login Quando preencher usuário com e-mail sem @ Então deve apontar a mensagem de erro no processo de Login | Passou |
| CT002.002 - login com e-mail válido | Dado que eu esteja na tela de login Quando preencher usuário com e-mail válido Então deve seguir com o processo de Login | Passou |
| CT002.003 - login com senha inválida | Dado que eu esteja na tela de login Quando o campo senha com uma senha inválida Então o sistema deve apontar senha incorreta | Passou |
| CT002.004 - login com senha válida | Dado que eu esteja na tela de login Quando o campo senha com uma senha válida Então o sistema deve permitir a continuidade de login com um e-mail e senha respectivamente corretos | Passou |
| 003-Navegação | CT003.001 - Fluxo de Navegação Carteira | Dado que eu esteja na tela de navegação da carteira Quando ao clicar no botão colunas e nos seus sub botões flegando ou desflegando Então o sistema deve permitir a variação do número de colunas da aplicação | Passou |
| CT003.002 - Fluxo de Navegação Estratégias | Dado que eu esteja na tela de navegação das estratégias Quando ao clicar no botão colunas e nos seus sub botões flegando ou desflegando Então o sistema deve permitir a variação do número de colunas da aplicação | Passou |
| Dado que eu esteja na tela de navegação das estratégias Quando ao clicar no botão adicionar estratégias o sistema deve permitir o preenchimento do formulário para cadastro de uma nova estratégia | Passou |
| CT003.003 - Fluxo de Navegação Trade | Dado que eu esteja na tela de navegação de trade Quando ao clicar no botão colunas e nos seus sub botões flegando ou desflegando Então o sistema deve permitir a variação do número de colunas da aplicação | Passou |
| Dado que eu esteja na tela de navegação de trade Quando ao clicar no botão Guia Rápido Então ele não deve funcionar por ainda não ter sido implementado | Bloqueado |
| Dado que eu esteja na tela de navegação de trade Quando ao clicar no botão Automações Então ele não deve funcionar por ainda não ter sido implementado | Bloqueado |
| Dado que eu esteja na tela de navegação de trade Quando ao clicar no botão Backtesting Então ele não deve funcionar por ainda não ter sido implementado | Bloqueado |
| Dado que eu esteja na tela de navegação de trade Quando ao clicar no botão Adicionar Automação Então ele não deve funcionar por ainda não ter sido implementado | Bloqueado |
| CT003.004 - Fluxo de Navegação Em andamento | Dado que eu esteja na tela de navegação do Em andamento Quando ao clicar no botão colunas e nos seus sub botões flegando ou desflegando Então o sistema deve permitir a variação do número de colunas da aplicação | Passou |
| CT003.005 - Fluxo de Navegação Histórico | Dado que eu esteja na tela de navegação de histórico Quando ao clicar no botão colunas e nos seus sub botões flegando ou desflegando Então o sistema deve permitir a variação do número de colunas da aplicação | Passou |

Códigos utilizados disponíveis no GIT: <https://github.com/RichardGuedesRib/tharseo_frontend_v2>

Vídeos evidenciando os testes (não anexado na tarefa devido ao tamanho do arquivo, então optamos por hospedar o vídeo no youtube e disponibilizar o link para acesso):

Teste de cadastro:

<https://youtu.be/nuhJqyoR0t0?si=7Voa40hMQuhY0EMl>

Teste de cadastro com e-mail já cadastrado:

<https://youtu.be/XSOwZFuERwY?si=2pw8wKCWDlvyGy61>

Teste de login:

<https://youtu.be/580YBzm9jUs?si=JXt27hG7mWkHEY49>

Teste de navegação Carteira:

<https://youtu.be/R_AblGRLfl8?si=sfAA8pHm1Jy8w-SW>

Teste de navegação Estratégias:

<https://youtu.be/13bQGqg8GyM?si=-uAuiesRk5njdJjK>

Teste de navegação Trade:

<https://youtu.be/L3z0P_y3n-8?si=qKsPqFRzoG2mNvPO>

Teste de navegação Em Andamento:

<https://youtu.be/PbhaihYdSXc?si=_i_1qDEiheCPxVKl>

Teste de navegação Histórico:

<https://youtu.be/CPkQPRwUZOg?si=0YzIl6I0xh81dcSv>