

EFICIÊNCIA E SEGURANÇA NA MALHA FERROVIÁRIA:
AUTOMAÇÃO DE TESTES NO NOVO SISTEMA DE DESPACHO DE
TRENS

Alexandre Sacoman, Jean Ferreira e Fernanda Malaquias

Resumo

Este artigo apresenta o desenvolvimento de um sistema automatizado de testes para um novo sistema de despacho de trens o TMDS (Train Management Dispatching System). A automação visa otimizar o processo de validação de cenários operacionais, como entrada e saída de via, licenças e posicionamentos, tradicionalmente realizados de forma manual por analistas. Utilizando a linguagem Python e o microframework Flask, o sistema permite ao usuário selecionar um cenário de teste por meio de uma interface web, que aciona rotinas específicas e executa macros automatizadas sobre o ambiente de simulação. O sistema é modular, baseado em scripts que leem arquivos de configuração *.json* e realizam ações simuladas via PyAutoGUI. Como resultado, obteve-se uma redução significativa no tempo de execução dos testes regressivos e maior confiabilidade nos resultados. A ferramenta permitiu que a equipe de testes focasse em casos mais complexos, contribuindo para o aumento da cobertura e da qualidade das validações.

Palavras-chave: automação de testes. TMDS. Python. macros. PyAutoGUI.

Abstract

This paper presents the development of an automated testing system for the new Train Management Dispatching System (TMDS). The automation aims to optimize the validation process of operational scenarios such as track entry and exit, licenses, and train positioning, which were traditionally performed manually by analysts. Using the Python programming language and the Flask microframework, the system enables users to select a test scenario through a web interface that triggers specific routines and executes automated macros within the simulation environment. The architecture is modular, based on scripts that read configuration files in .json format and perform simulated actions via PyAutoGUI. As a result, the regression test execution time was significantly reduced, and the reliability of outcomes was improved. The tool allowed the testing team to focus on more complex scenarios, increasing both coverage and validation quality

Sumário

Resumo	2
Abstract	3
1 Introdução	Erro! Indicador não definido.
2 Metodologia	6
3 Desenvolvimento e Resultados	8
3.1 Interface Web – Arquivo app.py	8
3.2 Processamento da Requisição – Arquivo tmds_operacao.py	9
3.3 Execução do Cenário – cen_XXX_nome_do_cenario.py	9
3.4 Execução das Macros – Arquivo funcoes_basicas.py	11
3.5 Registro e Análise por Logs – app.log	13
4 Conclusões	13
5 Referências Bibliográficas	14

1 Introdução

A evolução tecnológica no setor ferroviário tem impulsionado transformações significativas e a Rumo adquiriu um novo sistema de despacho de trens alinhada com esses objetivos. Nesse contexto, a automação de testes surgiu como uma estratégia indispensável para garantir a qualidade em um sistema crítico. Este trabalho apresenta a experiência de automação de testes aplicada ao sistema de despacho de trens. Dada a complexidade do software, que envolve múltiplos processos, como controle de entrada em via, licenças franca, SOS, permissiva e posicionamento de trens, os testes manuais mostraram-se ineficientes e onerosos. A ausência de automação comprometia a escalabilidade das validações e aumentava o risco de falhas em funcionalidades sensíveis à segurança e à operação ferroviária. Diante desse cenário, foi necessário adotar uma abordagem automatizada, capaz de validar rapidamente os componentes do sistema e permitir que os esforços humanos fossem direcionados a testes exploratórios e funcionalidades mais complexas. Essa solução não apenas aumentou a confiabilidade do processo de validação, como também reduziu significativamente o tempo de homologação de novas versões do software. Este artigo apresenta a metodologia empregada para estruturar a automação, os desafios enfrentados e os resultados obtidos, contribuindo com um estudo de caso aplicável a outros contextos ferroviários que demandem inovação tecnológica e eficiência.

2 Metodologia

A automação de testes do sistema TMDS foi desenvolvida com base em uma abordagem modular e escalável, orientada à frequência dos cenários operacionais nas mesas de controle. O processo metodológico compreendeu seis etapas principais: seleção de cenários, escolha da linguagem e biblioteca, definição da arquitetura modular, mapeamento de coordenadas, estruturação de servidores de controle e separação das responsabilidades de banco de dados.

A primeira etapa consistiu na seleção dos cenários de teste a serem automatizados. O critério adotado foi a repetitividade das operações nas mesas da operação ferroviária: Rondonópolis, Araraquara, Boa Vista, Santista e Portofer. A partir dessa análise, foram identificados 600 cenários prioritários, contemplando funcionalidades como entrada de via, posicionamento, licenças memorizadas, permissiva, SOS e interdições.

Em seguida, foi realizada a escolha da linguagem de programação e da biblioteca de automação. Optou-se pela linguagem Python, reconhecida por sua versatilidade e ampla adoção em tarefas de automação. A biblioteca PyAutoGUI foi selecionada por sua capacidade de simular interações com teclado e mouse, além de contar com documentação robusta e alta compatibilidade com sistemas desktop.

A estrutura da automação seguiu uma abordagem modular. O primeiro módulo estruturado foi o de simulação de macros, no qual foram mapeadas todas as coordenadas e parâmetros operacionais das macros utilizadas pelo simulador para envio no TMDS.

Na sequência, foi realizado o mapeamento das coordenadas de cada Seção de Bloqueio (SB) em todas as mesas. Para isso, foram criados arquivos no formato .json contendo o nome de cada SB e suas respectivas coordenadas X/Y na interface do TMDS. Esses dados foram utilizados pelos scripts de automação para localizar visualmente os elementos a serem acionados.

Para facilitar a utilização pelos analistas, foi implementado um servidor local em Flask, que fornece uma interface web para seleção da mesa e do cenário de teste. Essa solução proporcionou maior usabilidade e padronização do processo de disparo dos testes.

Por fim, visando garantir segurança e desacoplamento de responsabilidades, foi desenvolvido um servidor em Node.js responsável por todas as operações de leitura e

validação de dados no banco. Com isso, evitou-se a exposição de credenciais e conexões no código Python, promovendo baixo acoplamento entre os módulos e facilitando a manutenção futura do sistema.

Essa metodologia permitiu a construção de uma solução automatizada alinhada aos fluxos operacionais reais do sistema TMDS, com ganhos segurança e eficiência.

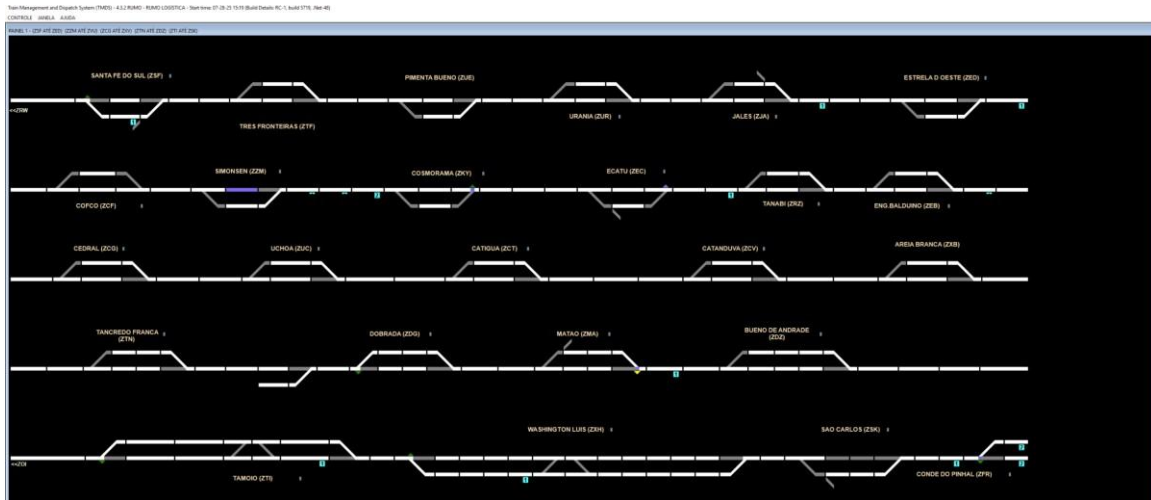


Figura 1 - Aplicação TMDS

```

{..} coordinate_sb.json M
tmnds_app > {..} coordinate_sb.json > ...
244      "ZTF415": ["495", "189"],
245      "ZTF": ["820", "215"],
246      "ZTF1": ["650", "189"],
247      "ZTF2": ["650", "149"],
248      "ZTFS": ["767", "189"],
249      "ZTFN": ["572", "189"],
250      "404ZTF": ["810", "189"],
251      "ZUE404": ["885", "189"],
252      "ZUE": ["1190", "121"],
253      "ZUE1": ["1040", "189"],
254      "ZUE2": ["1040", "230"],
    
```

Figura 2 - Arquivo de coordenadas das SBs

3 Desenvolvimento e Resultados

O sistema de automação de testes proposto é estruturado de forma modular, possibilitando a reutilização de componentes e a fácil manutenção de novos cenários de

teste. O processo completo inicia-se com a seleção do cenário pelo usuário e segue uma cadeia lógica de execução entre módulos, conforme descrito a seguir.

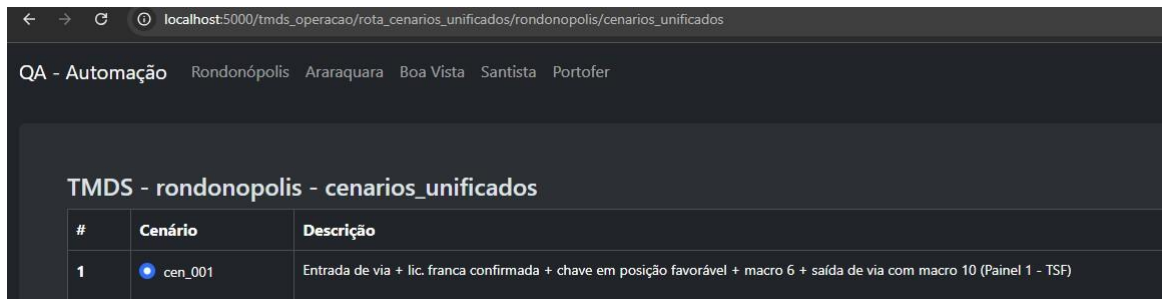
A entrada principal do sistema é realizada por meio do arquivo `app.py`, o qual executa um servidor local com interface Web (via Flask) que permite ao usuário selecionar qual cenário deseja executar. Essa interface disponibiliza todas as mesas configuradas no sistema: Rondonópolis, Araraquara, Boa Vista, Santista e Portofer. Cada mesa possui um conjunto de cenários associados, previamente definidos em arquivos `.json`.

Após a seleção do cenário, o `app.py` envia a requisição de execução para o arquivo `tmds_operacao.py`, responsável por orquestrar o processo. Esse módulo interpreta o cenário selecionado e direciona a execução para o respectivo script de cenário. Por exemplo, ao selecionar o cenário de entrada de via, o sistema invoca o script `cen_001_entrada_de_via.py`.

Cada script de cenário chama funções definidas no arquivo `funções_basicas.py` responsável pela implementação de comandos do PyAutoGui e a interface do simulador de macro e o TMDS.

3.1 Interface Web – Arquivo `app.py`

A execução da automação se inicia por meio do arquivo `app.py`, responsável por iniciar um servidor web utilizando o microframework Flask. Essa interface permite ao usuário selecionar, via navegador, qual cenário de teste deseja executar. A interface lista todos os casos de teste disponíveis para cada uma das mesas de teste atualmente suportadas.



#	Cenário	Descrição
1	<input checked="" type="radio"/> cen_001	Entrada de via + lic. franca confirmada + chave em posição favorável + macro 6 + saída de via com macro 10 (Painel 1 - TSF)

Figura 3 - Exemplo seleção de cenário

3.2 Processamento da Requisição – Arquivo `tmds_operacao.py`

Após o envio do cenário, a requisição é recebida pelo módulo `tmds_operacao.py`, que atua como orquestrador da automação. Esse arquivo contém funções responsáveis por analisar o cenário selecionado e direcioná-lo ao script correspondente. Por exemplo, ao receber um cenário de entrada de via padrão, o sistema executará a função `entrada_de_via_padrao()`, que por sua vez importa e executa o script específico.



```
tmds_operacao.py x
routes > tmds_operacao.py > fx rota_cenarios_unificados
34 def rota_cenarios_unificados(mesa, funcao):
86     elif cenario in [f"cen_{str(i).zfill(3)}" for i in range(50, 100)]:
87         if funcao == 'entrada_via':
88             entrada_de_via_padrao(mesa, funcao, cenario)
89         elif funcao == 'licenca_franca':
```

Figura 4 - Exemplo método entrada de via

3.3 Execução do Cenário – `cen_XXX_nome_do_cenario.py`

Cada cenário de teste possui seu próprio script localizado em arquivos separados (exemplo: `cen_001_entrada_de_via.py`). Esses scripts contêm funções com a lógica de negócio para processar os dados recebidos a partir de arquivos `.json`, que descrevem os parâmetros do teste, como locomotivas, prefixos, sentidos, estações, e posições iniciais. Esses scripts chamam funções definidas no módulo `funcoes_basicas.py`, onde se encontram os métodos de alto nível para interagir com os simuladores e com a interface do sistema TMDS.

```

{..} cen_001.json x
cenarios > araraquara > cenarios_unificados > {..} cen_001.json > ...
1  [
2  " __PASSO_1": " -> ENTRADA DE VIA + LICENÇA FRANCA",
3  "execucao_completa": "não",
4  "list_locos": ["8221"],
5  "list_locos_2": ["8223"],
6  "prefix": "I60",
7  "prefix_2": "L11",
8  "list_sbs": ["ZSF1"],
9  "list_authorized": ["sim"],
10 "list_sbs_lic": ["ZSF1", "ZTF415"],
11 "estender_licenca": "não",
12 "confirmar_licenca_franca": "sim",
13 "confirmar_licenca_franca_2": "não",
14 " __CONFIG_LICENCA_ESPECIAL__": " -> LICENÇA ESPECIAL",
15 "tipo_licenca_especial": "permissiva",
16 "loco_que_libera_permissiva": ["8221"],
17 "prefixo_que_libera_permissiva": "L11",
18 "sb_origem_lic_especial": "TSF732",
19 "sb_dest_lic_especial": "TSF1",
20 "km_sb_licenca_sos": "395.5",
21 "list_sbs_lic_2": ["TSF2", "TAL699"],
22 " __comment_": " -> CHAVES - Informar chave_mola ou chave_manual, SB, bloquear, interditar",
23 " __comment_": "Informar chave_mola ou chave_manual, SB, bloquear, interditar ou remover",
24 "manipular_chave_antes_licenca": "não",
25 "manipular_chave_depois_licenca": "não",
26 "tipo_chave": "chave_mola",
27 "sb_chave": ["TSF1S"],
28 "opcao_chave": "remover_interdicao",
29 "doble_licenca_com_41": "não"

```

Figura 5 - Parametrização de cenários

```

cen_001_entrada_de_via.py M x
cen_001_entrada_de_via.py > fx entrada_de_via_padrao
9 |
10 def entrada_de_via_padrao(mesa, diretorio, cenario):
11     # obtem os dados para EXECUTAR os cenários
12     config = massa_de_dados_json(mesa, diretorio, cenario)
13     try:
14         entrada_de_via_cbl(
15             config['list_locos'],
16             config['list_sbs'],
17             config['prefix'],)
18
19         entrada_de_via_tmnds(
20             config['list_locos'],
21             config['list_sbs'],
22             config['prefix'],
23             config['list_authorized'])
24
25         if config['list_authorized'] == 'sim':
26             saida_de_via(
27                 config['sb_saida_via'])
28     except Exception as e:
29         print(f"Erro capturado no fluxo principal: {e}")
30         print("Detalhes do erro:")
31         print(traceback.format_exc()) # Mostra a stack trace completa
32

```

Figura 6 - Método de entrada de via

3.4 Execução das Macros – Arquivo funcoes_basicas.py

O módulo funcoes_basicas.py centraliza as funções responsáveis pela execução das ações simuladas, tais como:

- entrada_de_via_cbl(): envia comandos de entrada de via para o simulador do de macros;
- entrada_de_via_tmnds(): interage diretamente com o TMDS simulando o clique e preenchimento dos campos;
- saida_de_via(), entre outros

Essas funções, por sua vez, invocam as classes Runner que encapsulam a lógica de automação utilizando a biblioteca PyAutoGUI.

```

funcoes_basicas.py x
funcoes_basicas.py > fx entrada_de_via_tmds
74 def entrada_de_via_tmds(list_locos, list_sbs, prefix, list_authorized):
80     list_sbs (string): SB para entrada de via
81     prefix (string): Prefixo do trem
82     list_authorized (string): Autoriza ou não a entrada de via
83
84     Returns:
85         int: ...
86     """
87     logger = get_logger('>> Function entrada_de_via_tmds')
88     logger.info('Starting entrada_de_via_tmds')
89     try:
90         time.sleep(8)
91         alt_tab('tmds') # Seleciona o TMDS
92
93         # ===== BANCO DE DADOS - MACRO 1 - VALIDAÇÃO =====
94         # Esse bloco faz a verificação na base de dados e traz o resultado da entrada de via
95         # Essa funcao retorna uma chave 'sucess' que pode ser true ou false
96         # Se retornar 'sucess == true' e o prefixo informado anteriormente é realizado o
97         result = valida_macros('in_archive', *list_locos, macro='1')
98         prefix_from_bd = result['prefixo']
99         # =====
100
101         if result['sucess'] and prefix == prefix_from_bd:
102             # ===== TMDS - ENTRADA DE VIA =====
103             # Chamada da função 'entrada_de_via' do TMDS com os dados informados anteriorm
104             tmds_ev = RunnerTmdsEntradaVia()
105             tmds_ev.entrada_de_via(list_sbs, list_authorized)
106             # =====
107

```

Figura 7 - Chamada das funções básicas

```

entrada_de_via.py x
tmds_app > entrada_de_via.py > EntradaDeVia > entrada_de_via
8 class EntradaDeVia:
15     self.logger = get_logger(self.__class__.__name__)
16
17     def entrada_de_via(self, sb, opcao):
18         # self.logger.info('entrada_de_via started method')
19
20         try:
21             self.tmds_sb.coordinates = sb # setter da classe TmdsCoordinates()
22             # getter da classe TmdsCoordinate()
23             x = self.region_tmds[0] + self.tmds_sb.coordinates[0]
24             y = self.region_tmds[1] + self.tmds_sb.coordinates[1]
25
26             # Posiciona o mouse no prefixo para entrada de via
27             self.automate.move_mouse(x, y)
28             # clica com o botão direito no prefixo para entrada de via
29             self.automate.click(button='right')
30             # Chama o método para selecionar a opção: autorizar/negar
31             self.menu_contexto_autorizar_negar()
32
33             if opcao == 'sim':
34
35                 self.permitir_entrada_via('sim')
36                 self.token.validar_token('sim') # Valida o token
37

```

Figura 8 - Exemplo simulação mouse e teclado para interação com o TMDS

3.5 Registro e Análise por Logs – app.log

Durante a execução dos testes automatizados, todas as ações realizadas são registradas em arquivos de log. Esse arquivo serve como fonte de verificação para auditar os passos realizados e confirmar se cada macro foi executada com sucesso. Esses registros são essenciais para depuração, validação técnica e rastreabilidade de resultados em auditorias internas.

```
----- STARTING NEW EXECUTION AT: 17/07/25 11h07 -----
2025-07-17 11:07:17,836 - werkzeug - WARNING - * Debugger is active!
2025-07-17 11:10:40,074 - >> Function entrada_de_via_cbl - INFO - Starting entrada_de_via_cbl
2025-07-17 11:10:41,076 - >> Function selection Window - INFO - Procurando por: Messagesimulator
2025-07-17 11:10:43,456 - >> Function selection Window - INFO - Janela 'Messagesimulator' ativada com sucesso.
2025-07-17 11:10:50,709 - >> Function selection Window - INFO - Procurando por: Train Management
2025-07-17 11:10:56,403 - >> Function entrada_de_via_tmms - INFO - Starting entrada_de_via_tmms
2025-07-17 11:11:04,411 - >> Function selection Window - INFO - Procurando por: Train Management
2025-07-17 11:11:09,823 - >> Function selection Window - INFO - Janela 'Train Management and Dispatch System (TMDS) - 4.3.1 RUMO - RUMO LOGÍSTICA
2025-07-17 11:11:09,855 - urllib3.connectionpool - DEBUG - Starting new HTTP connection (1): localhost:3000
2025-07-17 11:11:10,480 - urllib3.connectionpool - DEBUG - http://localhost:3000 "GET /tblmacrosinarchive?loco=8220&macro=1 HTTP/11" 200 263
2025-07-17 11:11:20,099 - EntradadeVia - INFO - # Class EntradadeVia - entrada_via_sobre_sb_licenciada method - não foi apresentado!
2025-07-17 11:11:23,044 - AutorizacaoOperacoes - INFO - # Class AutorizacaoOperacoes - validar_token method - não foi apresentado!
```

Figura 9 - Exemplo de LOG

4 Conclusões

A automação dos testes do sistema TMDS representou um avanço significativo no processo de validação de funcionalidades críticas para o despacho ferroviário. A adoção de uma abordagem modular, baseada em scripts em Python, integração com simuladores de macros e automação de interface gráfica com PyAutoGUI permitiu a construção de uma solução eficaz, escalável e adaptável às diversas mesas operacionais da malha ferroviária.

A aplicação da metodologia desenvolvida possibilitou a automatização de 600 cenários de teste, contemplando funcionalidades recorrentes e estratégicas do sistema. Como resultado, houve uma expressiva redução no tempo destinado ao planejamento e execução dos testes regressivos, diminuindo em aproximadamente uma semana o ciclo de homologação de cada versão do sistema. Além disso, a automação liberou os analistas para se concentrarem em validações mais complexas, contribuindo para uma melhoria qualitativa na análise dos processos ferroviários.

A estrutura de logs implementada, aliada à separação entre controle de execução e acesso ao banco de dados, garantiu rastreabilidade, segurança e padronização, alinhando-se às melhores práticas de desenvolvimento e operação de sistemas críticos.

Como continuidade ao trabalho, considera-se a possibilidade de expansão da suíte de testes para contemplar outros módulos do sistema TMDS, bem como a integração com pipelines de entrega contínua (CI/CD) e geração automática de relatórios. A experiência relatada neste estudo reforça o potencial da automação como ferramenta estratégica para ganho de eficiência, segurança operacional e qualidade no setor ferroviário.

5 Referências Bibliográficas

BSTQB – Brazilian Software Testing Qualifications Board. Syllabus CTFL 4.0 BR. Disponível em: http://bstqb.online/files/syllabus_ctfl_4.0br.pdf. Acesso em: 1 ago. 2025.

RUMO LOGÍSTICA. ET-RUM-PTC-00001_0Q – Modos de Operação CBL. Documento técnico interno. 2023.