Documento de Apresentação dos Testes Automatizados – Backend Vidya Force (Versão 2)

1. **Introdução**

Este documento apresenta, em linguagem simples e objetiva, o que é verificado pelo script de testes automatizados da Versão 2 aplicado ao Backend Vidya Force. O objetivo é que qualquer pessoa, mesmo sem conhecimento técnico aprofundado, consiga entender o propósito dos testes e como eles ajudam a garantir a qualidade do sistema.

2. O que é o script de testes da Versão 2

O script de testes é um conjunto de verificações automáticas executado junto com as requisições da coleção "Backend - Vidya Force" no Postman. Ele observa as respostas da API e confere se elas estão dentro do que o sistema espera em termos de funcionamento, formato das informações, regras de negócio, desempenho e segurança.

A Versão 2 utiliza um único script configurado no nível da collection. Isso significa que, ao rodar a collection, todas as chamadas são avaliadas automaticamente, sem precisar configurar teste por teste manualmente.

3. Visão geral do que é testado

De forma resumida, o script de testes da Versão 2 verifica os seguintes pontos principais:

- Se os serviços estão estáveis, sem erros graves de servidor.
- Se as respostas seguem o formato combinado (campos e estruturas corretas).
- Se as informações essenciais para o negócio estão presentes (por exemplo, identificadores, códigos de parceiros, dados de pedidos).
- Se as principais jornadas funcionam, como login e operações de pedidos.
- Se as mensagens de erro são claras e padronizadas quando algo dá errado.
- Se os tempos de resposta estão dentro de limites aceitáveis (SLA).
- Se dados sensíveis não estão sendo expostos de forma indevida.
- Se alguns dados cadastrais básicos têm qualidade mínima (como UF, CEP e documentos).

4. Principais categorias de testes

4.1 Estabilidade e disponibilidade dos serviços

Garante que as APIs respondam sem erros críticos de servidor (códigos 5xx) em cenários considerados positivos. Isso ajuda a identificar falhas de infraestrutura e problemas internos do sistema.

4.2 Formato correto das respostas (Contrato)

Verifica se as respostas seguem o padrão esperado, por exemplo: uso correto do campo hasError, presença de mensagens de erro quando necessário, consistência entre a quantidade informada e os dados retornados e ausência de conteúdo HTML indevido em respostas JSON.

4.3 Regras de negócio básicas

Confere se informações essenciais do negócio estão presentes e coerentes, como pedidos com identificador e cliente, parceiros com código válido, produtos com código e descrição, e outras regras simples que garantem que os dados retornados façam sentido para o uso diário.

4.4 Funcionamento de fluxos importantes

Acompanha fluxos chave, como o login e o ciclo dos pedidos. Por exemplo, verifica se o login retorna dados suficientes para autenticação e se operações de pedido retornam um número de pedido que permita a continuidade das próximas etapas.

4.5 Tratamento adequado de erros

Quando ocorre um erro, o script valida se a API responde de forma clara e padronizada, informando que houve falha e trazendo uma mensagem compreensível, em vez de retornar respostas genéricas ou silenciosas.

4.6 Desempenho (SLA)

Mede o tempo de resposta das chamadas. Existe um limite geral para todas as requisições e limites mais rígidos para funcionalidades críticas, como login e dashboard. Essa medição ajuda a garantir uma experiência ágil para o usuário.

4.7 Segurança das informações

Verifica se não há exposição de informações sensíveis, como senhas ou segredos técnicos, e se chamadas autenticadas não recebem respostas incorretas de não autorização sem justificativa.

4.8 Qualidade de dados básicos

Aponta possíveis problemas simples de cadastro, como UF inválida, CEP em formato incorreto ou documentos possivelmente duplicados na mesma resposta. Esses alertas ajudam na melhoria contínua da base de dados.

5. Benefícios para o negócio

A utilização deste script de testes automatizados traz segurança e visibilidade para o time técnico e para as áreas de negócio. Em resumo, ele ajuda a garantir que as funcionalidades críticas estejam funcionando conforme o esperado, que mudanças não quebrem regras importantes sem serem percebidas, que erros sejam identificados mais cedo e que o ambiente ofereça uma experiência mais estável, segura e confiável para usuários e clientes.

6. O que mudou na Versão 3 (Add-on)

Esta atualização não substitui a Versão 2 — ela a complementa. Abaixo estão as melhorias incluídas na Versão 3 (Add-on), escritas de forma simples e objetiva:

• Pre-request global: injeta o header accessData automaticamente (quando definido no ambiente) e garante Authorization Basic gerado a partir de usuário e senha, evitando falhas por falta de cabeçalhos.

• Binários: valida se PDFs, DANFE, Boleto e Imagens possuem MIME correto e um tamanho mínimo (evita aceitar HTML de erro como PDF, por exemplo).

• Paginação: impede repetição de itens entre páginas e confere coerência do número de página (ex.: /ppid/getPrices?page=N).

• Invariantes cross-endpoints:

– /partner/save → o parceiro precisa aparecer em /partner/list.

– /ppid/{nunota}/saveAttachment → deve refletir em /ppid/{nunota}/listAttachment.

– /user/{id}/changePhoto → precisa refletir em /user/{id}/imagem (tipo de arquivo e tamanho).

• Idempotência: repetir operações críticas (como confirmarPedido/excluirItemPedido) deve resultar em erro controlado, nunca 'sucesso silencioso'.

• Negativos padronizados: sufixos no nome do request como [NEGATIVO], [SEM AUTH], [SEM ACCESSDATA], [ID INEXISTENTE] disparam automaticamente expectativas de 400/401/403/404 e exigem mensagem clara no JSON de erro.

• Segurança: verifica se o JSON não vaza dados sensíveis (ex.: password/senha/secret).

• Reset real das flags: o request “00 – [RESET FLAGS]” agora limpa as variáveis skip_* e caches auxiliares (como a lista de itens já vistos na paginação).

7. Como executar após a atualização

1) Execute primeiro o request “00 – [RESET FLAGS]” para limpar flags e caches de execução anteriores.

2) Rode a collection normalmente (Runner ou Newman).

3) Para cenários negativos, crie clones dos requests e adicione sufixos no nome (ex.: [NEGATIVO][SEM AUTH]) para ativar automaticamente as validações negativas.

4) Verifique os resultados no painel de testes do Postman — as novas asserções aparecem junto às da Versão 2.

8. Boas práticas e segurança

• Não versionar credenciais: mantenha username/password e accessData em Environments seguros (não comite valores reais).

• Reset entre suítes: antes de suítes completas, execute o reset para garantir que nenhuma flag de 'skip' interfira no resultado.

• Idempotência/negativos: mantenha os clones negativos e revise mensagens de erro; elas são parte do contrato para o front-end.

9. Refinamentos aplicados (V3.1)

- Binários condicionados: asserções de PDF/DANFE/Boleto e Imagens executam apenas em respostas 2xx e não-JSON (evita conflito com erros JSON).
- Whitelist de segurança (auth/token): em rotas de autenticação, campos auth/token são aceitos; fora delas, seguem bloqueados, além de password/senha/secret.
- Paginação namespaced por vendedor: controle de itens já vistos inclui o codVend, evitando falsos positivos de repetição.
- Idempotência robusta: replays de mutações clonam o corpo em raw, x-www-form-urlencoded e form-data.
- Obrigatoriedade de accessData: rotas /ppid/ cobertas por teste dedicado de presença do header accessData.
- Checks leves adicionais: validações mínimas para mensagens (/ppid/message) e solicitações de entrega (/ppid/solicitacoesEntrega).

10. Refinamentos aplicados (V3.2)

- Binários blindados: PDFs/DANFE/Boleto validam assinatura %PDF- e tamanho mínimo; imagens checam MIME e tamanho. Asserções executam apenas quando a resposta não é JSON, evitando falso-positivo em erros estruturados.
- Whitelist de segurança (auth/token): campos de autenticação são aceitos apenas em rotas de login/refresh; fora delas seguem bloqueados, assim como password/senha/secret.
- Paginação namespaced por vendedor: controle de itens já vistos inclui codVend, adicionando verificação de coerência entre query.page e body.page e caso de borda [PAGE OUT OF RANGE].
- Idempotência robusta: mantidos replays de mutações; reforço para que o 2º envio retorne erro controlado (4xx/hasError).
- Obrigatoriedade de accessData: rotas /ppid/ cobertas por teste dedicado; Pre-request corrige typos (accesData → accessData) e injeta o header quando configurado.
- Checks leves adicionais: validação de headers por domínio e SLA leve (<3s) em operações críticas (ex.: geração de PDFs).

11. Impacto prático

- Menos falsos positivos em arquivos binários e fluxos de autenticação.
- Maior previsibilidade em paginação (sem repetição entre páginas e com “fora do alcance” controlado).
- Contrato mais rígido em rotas /ppid/ com accessData obrigatório.
- Operação multi-cliente/ambiente apenas trocando variáveis, sem editar requests.

12. Refinamentos aplicados (V3.2.1)

- Bootstrap reset-once: limpeza automática de resíduos entre rodadas (ex.: skip_*, v3_*, seen_*, idempotency_*, page_state*, run_*, tmp_, last_*), preservando credenciais e baseUrl. Gera run_id único por execução.
- Testes sempre-registrados: binários, paginação, headers /ppid/, SLA e negativos passam a registrar pm.test mesmo quando o cenário não se aplica, marcando “N/A”. Mantém a contagem de testes estável entre runs.

- Paginação coerente: se a API informa body.page/pagina, comparamos com query.page (quando houver), sem alterar coberturas já existentes.
- Boas práticas de Runner: recomenda-se Persist variables = OFF e iniciar pela requisição “00 – [RESET FLAGS]”. O bootstrap ajuda mesmo quando o reset é esquecido.
- Compatibilidade total: incrementos não removem o que já existia (V2/V3). Quando já coberto, o guard retorna “N/A” em vez de falhar.

13. Impacto prático

- Contagem de testes previsível entre a 1ª e a 2ª bateria (reduz variações causadas por estado residual).
- Menos ruído em binários/paginação (asserções só aplicam quando o cenário é válido; caso contrário, “N/A”).
- Execuções mais confiáveis em CI (menor risco de quedas artificiais de cobertura).
- Sem quebra de fluxo existente: mantém V2/V3 e apenas adiciona proteção e estabilidade.

14. Refinamentos aplicados (V3.3)

• JWT (claims mínimos): em fluxos de autenticação, valida presença de `exp` e expiração ≥ 5 min.

• Observabilidade: exige `X-Request-ID`/`X-Correlation-ID`/`traceparent` em respostas.

• Rate limit: quando presentes, verifica `Remaining ≤ Limit`.

• Timestamps ISO-8601/UTC: campos de data padronizados (evita problemas de fuso).

• Content-Type para JSON: se o corpo *parece* JSON, exige `application/json`.

• Cookies seguros: quando há `Set-Cookie`, exige flags `Secure` e `HttpOnly`.

• Consistência monetária: soma de itens ≈ total (tolerância 0,05) em detalhes de pedido.

• Baseline de schema: salva conjunto de chaves por rota/método e alerta drift.

• Cache HTTP: em GET 2xx, exige `ETag` ou `Last-Modified` (revalidação opcional).

• SLA percentis: coleta `responseTime` para cálculo de p95/p99 em request de relatório.

• Paginação — limites: `pageSize` dentro de 1..200 e retorno como array.

• Uploads: filename ecoado sem `../`/`/`/`\` e tamanho 1..255.

15. Impacto prático (V3.3)

• Mais robustez de contrato (schema drift) e previsibilidade para clientes.

• Redução de bugs de fuso/serialização (datas ISO-UTC) e cabeçalhos coerentes (Content-Type/RateLimit).

• Melhor rastreabilidade em produção (Correlation/Trace ID) e segurança (cookies).

• Medição de performance mais útil (coleta para p95/p99) e cache HTTP bem configurado.

• Menos inconsistências financeiras (itens vs total) e paginação dentro de limites controlados.

• Maior segurança e higiene em uploads (filename).