Mapeamento e teste dos requests (API PetStore)

Squad 3 - CapsLock
Andressa Mota
Guilherme Taveira
Marcelo Ferreira
Maycon Douglas
Willian

1. Objetivo

O propósito deste relatório é verificar se a API PetStore atende aos contratos e cenários definidos na documentação Swagger, garantindo a robustez, conformidade e estabilidade dos endpoints principais.Focamos em validar operações CRUD, fluxos de autenticação e manipulação de dados, identificando falhas de conformidade e comportamentos inesperados.

2. Escopo dos Testes

- Cadastro, consulta, atualização e remoção de Pets
- Geração e consulta de pedidos (Orders)
- Cadastro, autenticação e gerenciamento de usuários
- Validação de autenticação por token (Auth)
- Verificação de códigos de status HTTP e conformidade de esquemas JSON

3. Ferramentas Utilizadas

- Swagger UI 3.x : Para consultas interativas e exportação do JSON/YAML da API para referência.
- Postman v10.9.0 : Para criação de Collection com todos os endpoints e variáveis de ambiente (baseUrl, apiKey).

4. Metodologia de Teste

Estratégia Utilizada: Testes Baseados em Especificação (Specification Testing)
Justificativa: permite validar rigorosamente cada campo e comportamento esperado, detectando inconsistências entre implementação e documentação.

Heurísticas Aplicadas:

- Verificação de códigos de erro
- Consistência de esquema JSON (tipos, obrigatoriedade)
- Tratamento de dados inválidos e limites de input
- Autenticação: validade e expiração de tokens

5. Técnica de Execução

Execução Orientada por Collection no Postman:

1. Configuração dos ambientes (dev, staging) com variáveis globais.
2. Importação da Collection gerada a partir do Swagger.
3. Execução sequencial dos requests em modo Runner, com gravação de logs.
4. Uso de scripts “Tests” para assertivas de status code.

6. Mapeamento de Endpoints e Casos de Teste

- **ENDPOINTS /PET**

Endpoint	Método	Descrição	Retorno do Teste	Resultado Esperado
/pet	POST	Adiciona novo pet	`{ "id": 101, "name": "Rex", "status": "available" }`	200 OK, JSON com dados gravados Deveria retornar 201 no Status Code
/pet	PUT	Atualiza dados de pet	`{ "id": 101, "name": "Rexy", "status": "sold" }`	200 OK, JSON atualizado

/pet/{petId}	DELETE	Remove pet	Pet deleted	200 OK, mensagem de sucesso
/pet/findByStatus	GET	Consulta todos Pets por status	`{"id": 27555, "category": {"id": 1, "name": "Dogs"}, "name": "Fluffy","photoUrls": [ "https://example.com/photo.jpg".....`	200 OK, JSON retornando consulta todos os Pets por status
/pet/findByTags	GET	Consulta Pets por tags	`"tags": [ { "id": 1, "name": "cute"}],"status": "sold" },`	200 ok, JSOn retornando consulta por tags

- **ENDPOINTS /STORE**

Endpoint	Método	Descrição	Retorno do Teste	Resultado Esperado
/store/inventory	GET	Retorna contagem de pets por status	—	200 OK, JSON: `{ "available": 5, ... }`
/store/order	POST	Cria novo pedido	`{ "id": 201, "petId": 101, "quantity":`	200 OK, JSON do pedido criado. Deveria

			`1, "status": "placed" }`	retornar 201 no Status Code
/store/order/{orderId}	GET	Consulta pedido por ID	—	200 OK, JSON com o pedido
/store/order/{orderId}	DELETE	Cancela pedido	—	200 OK, confirmação de exclusão

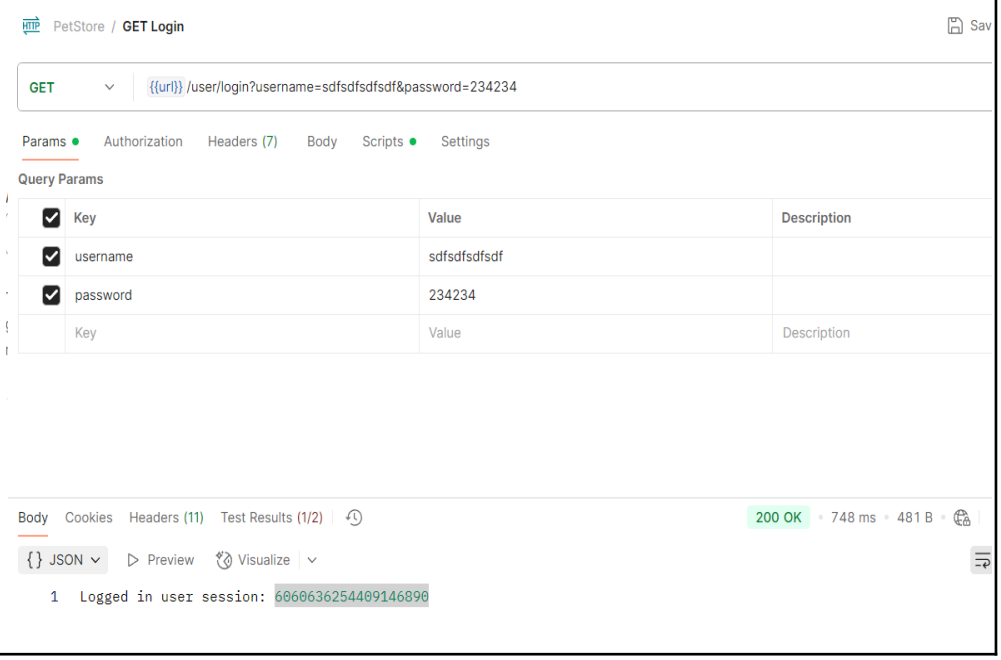
- **ENDPOINTS /USER**

Endpoint	Método	Descrição	Retorno do Teste	Resultado Esperado
/user	POST	Cria usuário	`{ "id": 301, "username": "andressa", "firstName": "Andressa", "email": "a@example.com" }`	200 OK, JSON do usuário criado Deveria retornar 201 no Status Code

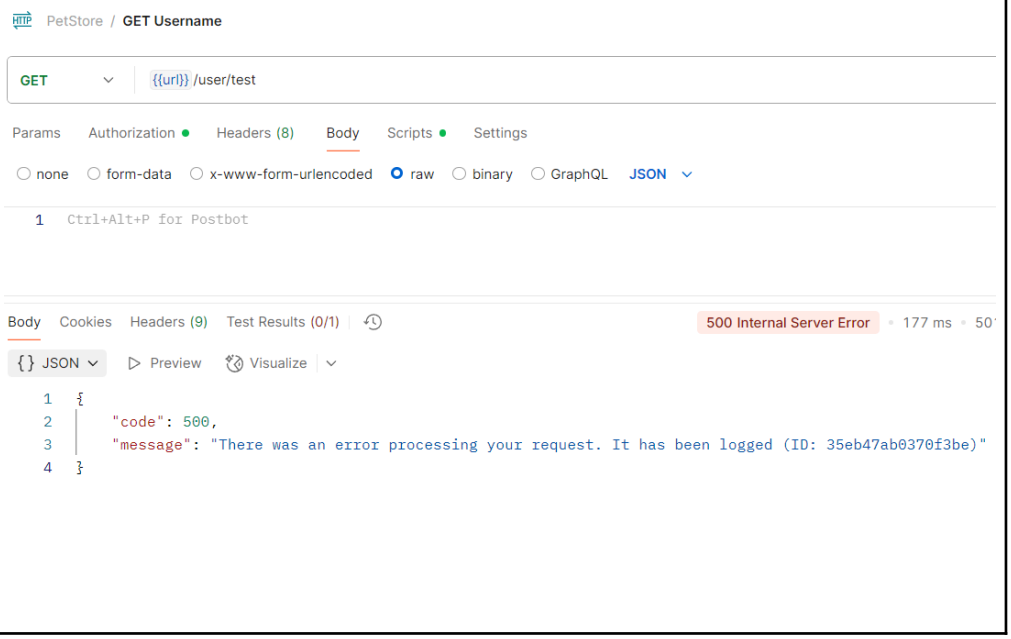
/user/createWithArray	POST	Cria múltiplos usuários (array)	`[ { "id": 302, "username": "maria" }, { "id": 303, "username": "joao" } ]`	200 OK, lista de usuários criados
/user/createWithList	POST	Cria múltiplos usuários (lista)	*mesma estrutura de array acima*	200 OK, lista de usuários criados
/user/login	GET	Autentica usuário	`Logged in user session: 1748902251825163146`	200 OK, token em `X-API-KEY`
/user/logout	GET	Encerra sessão do usuário	—	200 OK, mensagem de logout
/user/{username}	GET	Consulta dados de um usuário por username	—	200 OK, JSON com dados do usuário
/user/{username}	PUT	Atualiza dados de um usuário	`{ "id": 301, "username": "andressa", "firstName": "Andre", "email": "a@novo.com" }`	200 OK, JSON com dados atualizados
/user/{username}	DELETE	Remove usuário	—	200 OK, confirmação de exclusão

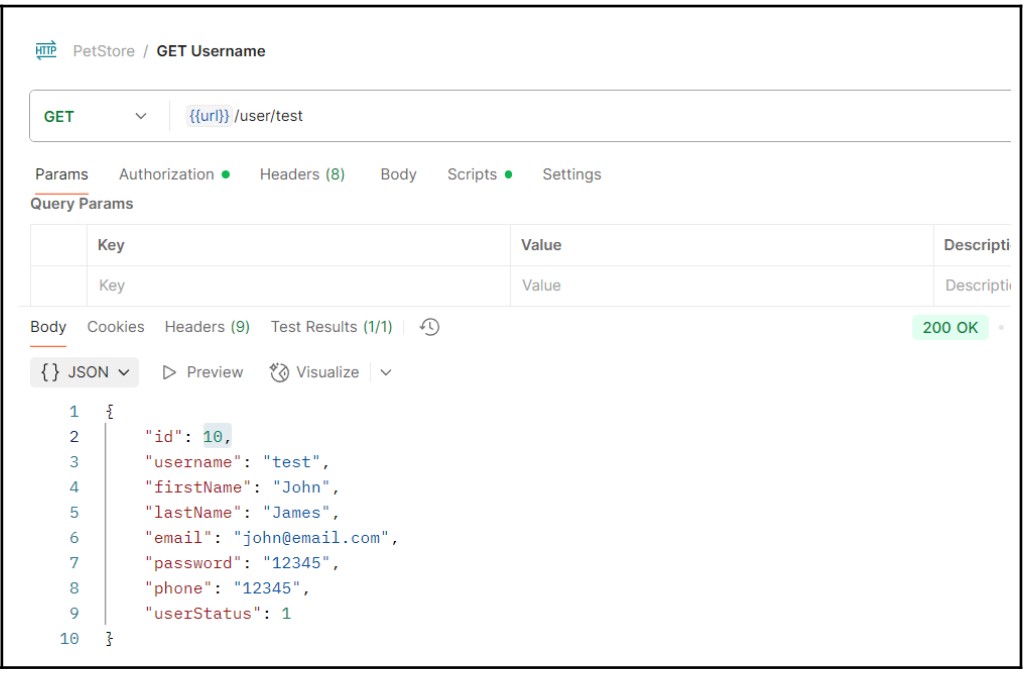
7. Registro dos resultados

GET user/login

Data e hora	07/08/2025 às 15:30
testador	Squad CapsLock
Funcionalidade explorada	GET /user/login
Observações	O endpoint /user/login aceita qualquer valor para username e password
Melhorias sugeridas	Se faz necessário a criação de diferentes formas de autenticação de usuário e senha, como por exemplo o uso de api-key ou token.
Evidências	

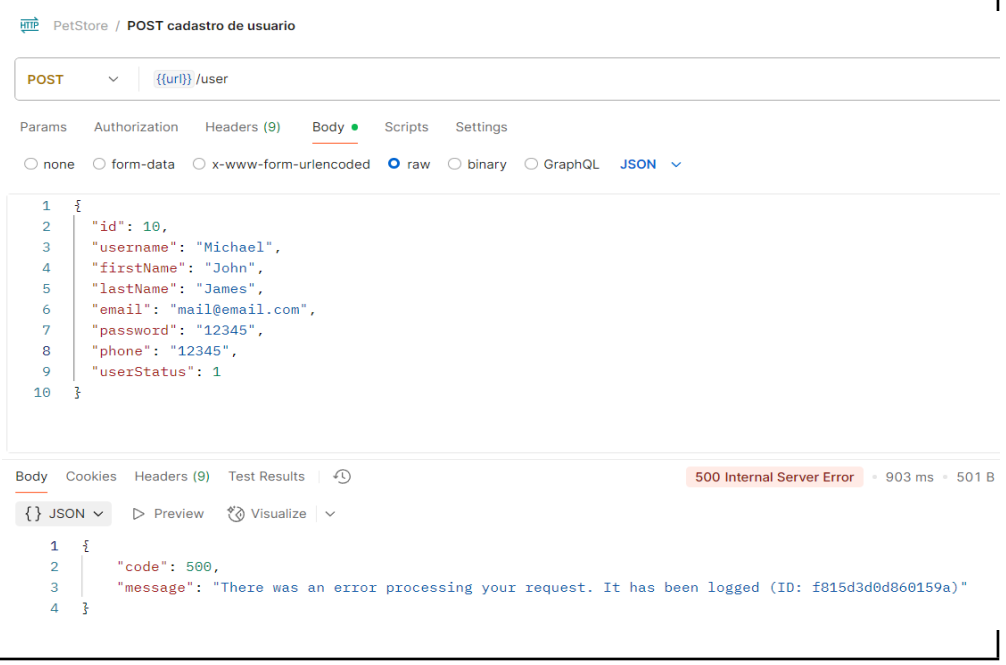
GET /user/{username}

Data e hora	07/08/2025 às 15:42
testador	Squad CapsLock
Funcionalidade explorada	GET /user/{username}
Observações	/user/{username} está com instabilidade no servidor e nos primeiros testes retornou com erro 500. Hoje (08/08/2025) está funcionando como mostra a imagem capturada abaixo.
Melhorias sugeridas	Se faz necessário conferir o motivo da instabilidade do servidor de api.
Evidências	

POST /user

Data e hora	07/08/2025 às 15:58
testador	Squad CapsLock
Funcionalidade explorada	Post /user
Observações	Post /user não está cadastrando usuário e retornando erro 500.

Melhorias sugeridas	Se faz necessário reparar a função de cadastro de usuário
Evidências	

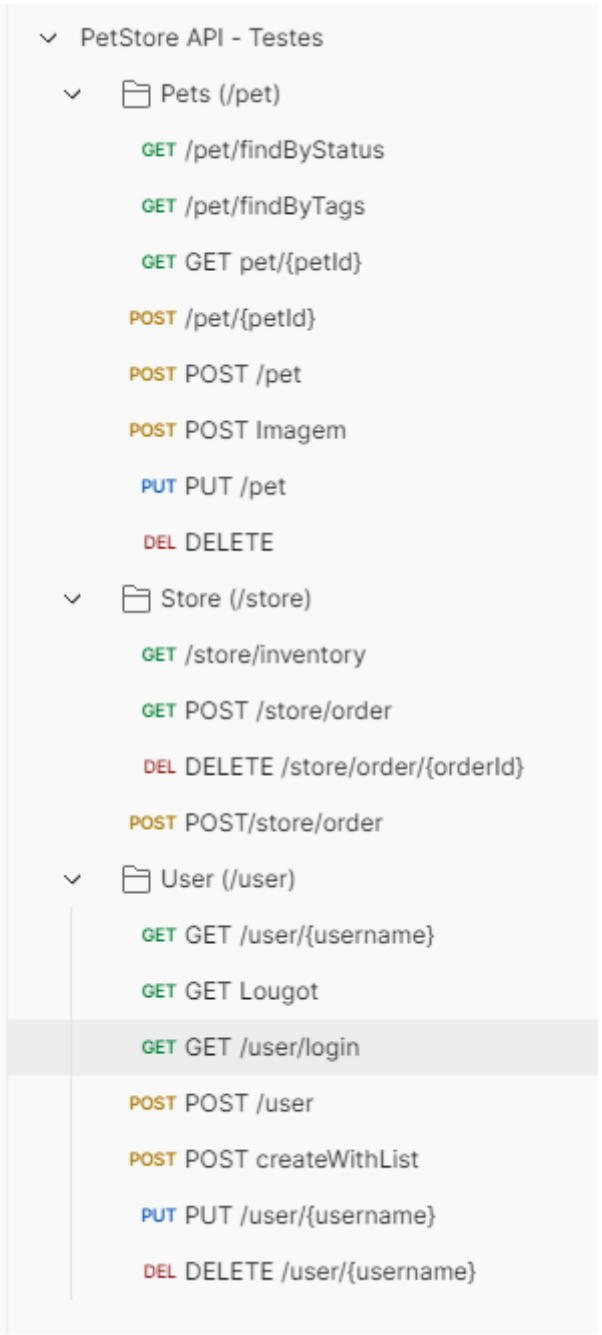
8. Resultados e Observações

- Todos os endpoints CRUD de Pet e Store responderam conforme o esquema descrito, sem desvios de tipo ou dados faltantes.
- Alguns endpoints de User apresentam erros ou comportamentos não esperados.

9. Conclusão

Com a identificação de alguns bugs nos testes dos endpoints, temos que, embora a base funcional esteja em funcionamento, há pontos críticos que impedem a estabilidade e a confiabilidade da API em um ambiente de produção.

10. Collection Postman


PetStore API - Testes
Pets (/pet)
GET /pet/findByStatus
GET /pet/findByTags
GET GET pet/{petId}
POST /pet/{petId}
POST POST /pet
POST POST Imagem
PUT PUT /pet
DEL DELETE
Store (/store)
GET /store/inventory
GET POST /store/order
DEL DELETE /store/order/{orderId}
POST POST/store/order
User (/user)
GET GET /user/{username}
GET GET Lougot
GET GET /user/login
POST POST /user
POST POST createWithList
PUT PUT /user/{username}
DEL DELETE /user/{username}