Roteiro de Testes: Cadastro de Serviço Pet (API + Integração de Dados)

Objetivo: Validar a funcionalidade completa de CRUD do serviço pet, garantindo a integridade dos dados e o correto funcionamento da API.

Ferramentas: Postman (para requisições HTTP e validações de API) e **Cliente SQL** (para validação do Banco de Dados).

Endpoint Base: {{base_url}}/api/servicos

FASE 1: Testes Funcionais da API (CRUD Básico) com Postman

Esta fase foca em garantir que a API responda corretamente às operações básicas (CRUD).

ID Teste	Operação	Método	Endpoint	Condição / Dados de Entrada	Validação no Postman (Status e Body)	Ação de Seguimento
1.1	**CREATE**	POST	/api/servicos	**Dados Válidos:** nome: "Banho Simples", preco: 50.00, duracaoMinutos: 30,	Status: **201 Created**. O body da resposta deve conter o objeto criado com	Salvar o idServico em uma variável de ambiente.

				idCategoria: 1	um idServico gerado.	
1.2	**READ**	GET	/api/servicos/{{idServico}}	Usar o idServico salvo no Teste 1.1.	Status: **200 OK**. O body deve corresponder exatamente aos dados enviados no Teste 1.1.	N/A
1.3	**UPDATE**	PUT	/api/servicos/{{idServico}}	**Alterar um campo:** Mudar preco de 50.00 para 65.00 e duracaoMinutos para 45.	Status: **200 OK**. O body deve refletir as novas alterações.	N/A
1.4	**READ (Após Update)**	GET	/api/servicos/{{idServico}}	Usar o idServico salvo.	Status: **200 OK**. O body deve confirmar o novo preco (65.00) e duracaoMinutos (45).	N/A
1.5	**DELETE**	DELETE	/api/servicos/{{idServico}}	Usar o idServico salvo.	Status: **204 No Content** ou **200 OK** (depende da API, mas sem corpo de resposta).	N/A

1.6	**READ (Após Delete)**	GET	/api/servicos/{{idServico}}	Usar o idServico que foi excluído.	Status: **404 Not Found** ou mensagem de erro apropriada.	N/A

FASE 2: Testes de Regras de Negócio e Validação (API) com Postman

Esta fase testa as regras de validação (campos obrigatórios, unicidade e tipos de dados) conforme definido no Diagrama de Classes UML.

ID Teste	Operação	Método	Endpoint	Condição / Dados de Entrada	Validação no Postman (Status e Body)	Regra de Negócio Testada
2.1	**CREATE**	POST	/api/servicos	**Campo Obrigatório Ausente:** Omitir o campo nome.	Status: **400 Bad Request**. A mensagem de erro deve indicar que o campo nome é obrigatório.	Validação de Campo Obrigatório.
2.2	**CREATE**	POST	/api/servicos	**Campo Obrigatório Ausente:** Omitir o campo preco.	Status: **400 Bad Request**. A mensagem de erro deve indicar que o campo preco é obrigatório.	Validação de Campo Obrigatório.

2.3	**CREATE**	POST	/api/servicos	**Unicidade (Chave Única):** Tentar criar um serviço com o nome "Banho Simples" (se já existir).	Status: **409 Conflict** ou **400 Bad Request**. A mensagem deve indicar duplicidade de nome.	Validação de Unicidade.
2.4	**CREATE**	POST	/api/servicos	**Tipo de Dado Inválido:** Enviar o preco como String (ex: "cinquenta").	Status: **400 Bad Request**. A mensagem de erro deve indicar um erro de tipo de dado (Decimal/Currency).	Validação de Tipo de Dados.
2.5	**CREATE**	POST	/api/servicos	**FK Inexistente:** Usar um idCategoria que não existe no banco (ex: 999).	Status: **400 Bad Request** ou **500 Internal Server Error** (com tratamento). A mensagem deve indicar erro de chave estrangeira.	Validação de Chave Estrangeira (FK).

2.6	**UPDATE**	PUT	/api/servicos/{{idServico}}	**Unicidade no Update:** Tentar mudar o nome de um serviço existente para um nome que já é usado por outro serviço.	Status: **409 Conflict** ou **400 Bad Request**.	Validação de Unicidade no Update.

FASE 3: Testes de Integração e Persistência com SQL

Esta fase garante que as operações da API estejam refletindo corretamente no Banco de Dados, validando não apenas o dado principal, mas também os campos de controle e relacionamentos (FK).

ID Teste	Operação Prévia (API)	Ação de Teste (SQL)	Query SQL Exemplo	Resultado Esperado (BD)
3.1	**CREATE** (Teste 1.1)	Inserir um novo registro via API e verificar se está na tabela.	SELECT nome, preco, idCategoria FROM Servico WHERE idServico = {{id_inserido}};	O registro deve ser encontrado, e os campos nome, preco e idCategoria devem corresponder exatamente aos dados da requisição.
3.2	**CREATE** (Teste 1.1)	Verificar se os campos de controle (Auditoria) foram preenchidos.	SELECT dataCriacao, idUsuarioCadastro FROM Servico WHERE idServico = {{id_inserido}};	Os campos dataCriacao e idUsuarioCadastro (FK para UsuarioAdmin) devem estar preenchidos com valores válidos.

3.3	**UPDATE** (Teste 1.3)	Alterar um registro via API e verificar a alteração na tabela.	SELECT preco, duracaoMinutos FROM Servico WHERE idServico = {{id_atualizado}};	Os campos devem refletir os novos valores (preco: 65.00, duracaoMinutos: 45).
3.4	**DELETE** (Teste 1.5)	Excluir um registro via API e verificar se ele foi removido do banco.	SELECT * FROM Servico WHERE idServico = {{id_deletado}};	A query não deve retornar **nenhum registro**.
3.5	**Regra FK** (Teste 2.5)	Tentar inserir diretamente no BD um serviço com um idCategoria inexistente.	INSERT INTO Servico (nome, preco, idCategoria) VALUES ('Erro FK', 10.00, 999);	O SGBD deve retornar um **erro de violação de chave estrangeira**.

4. Pré-Requisitos e Configuração

1. **Ambiente:** A API deve estar ativa e acessível na URL {{base_url}}.
2. **Dados de Teste:** É necessário que a tabela CategoriaServico tenha pelo menos uma categoria válida cadastrada (ex: idCategoria = 1).
3. **Variáveis Postman:** Criar uma Collection no Postman e definir variáveis de ambiente essenciais, como {{base_url}} (URL da API) e {{idServico}} (para encadear os testes de CRUD).
4. **Autenticação:** Se a API exigir autenticação (JWT, Basic Auth, etc.), o *token* de acesso deve ser configurado no *Header* de todas as requisições, geralmente na aba "Authorization" do Postman.