4 – API – SISTEMA DE AGENDAMENTO PARA CASTRAÇÃO ANIMAL

Neste capítulo, apresenta-se uma proposta de trabalho interdisciplinar, tem como objetivo o desenvolvimento de uma API destinada a atender às necessidades de uma instituição pública e sem fins lucrativos da cidade de Avanhandava/SP. A proposta surgiu como exercício prático de integração entre diferentes áreas do conhecimento, o que possibilitou a aplicação de conceitos técnicos de programação, na disciplina de Programação *Web,* ministrada pelo professor Dr. Anderson Pazin, no quarto bimestre. Além de fortalecer a compreensão dos conteúdos teóricos estudados, o trabalho buscou demonstrar a relevância social da tecnologia quando direcionada a soluções que apoiam o interesse coletivo.

De forma geral, uma API é “um conjunto de padrões, ferramentas e protocolos que permite a criação mais simplificada e segura de plataformas, pois permite a integração e a comunicação de *softwares* e seus componentes” (ALURA, 2025).

A IDE utilizada para escrever, compilar e testar o código da API foi o *IntelliJ IDEA.*

Já a linguagem de programação utilizada foi o *Java*. Pois permite a criação de sistemas escaláveis e seguro, sendo adequado para o desenvolvimento de *APIs*.

A aplicação desenvolvida utiliza *Spring Boot* como *framework* para a construção da API, sendo configurada por meio do arquivo “application.properties” sendo definido os parâmetros essenciais para o funcionamento da aplicação do servidor web e do banco de dados.

O nome da aplicação foi definido como “castracaoAnimal”, a porta do servidor web foi configurada para “8081” e o contexto base da API foi definido como “/castracaoanimal”, permitindo organizar todas as todas sob um caminho comum.

Para o armazenamento de dados, foi utilizado o **banco H2**, um banco de dados em memória com suporte à persistência em arquivo local. As configurações contemplam a *Uniform Resource Locator* (URL) de conexão, o driver *Java Database Connectivity* (JDBC) e as credenciais de acesso. Adicionalmente, o console *web* do H2 foi habilitado, de modo a permitir a visualização e manipulação dos dados diretamente pelo navegador.

O **Hibernate/JPA** (*Java Persistence API* – Interface de Persistência Java) foi configurado para utilizar o dialeto H2 e manter o banco atualizado automaticamente (`ddl-auto=update`), o que garante maior agilidade no desenvolvimento e evitando inconsistências de esquema, com base nas entidades na aplicação, garantindo que a estrutura de banco seja criada e adaptada conforme necessário sem a perda de dados.

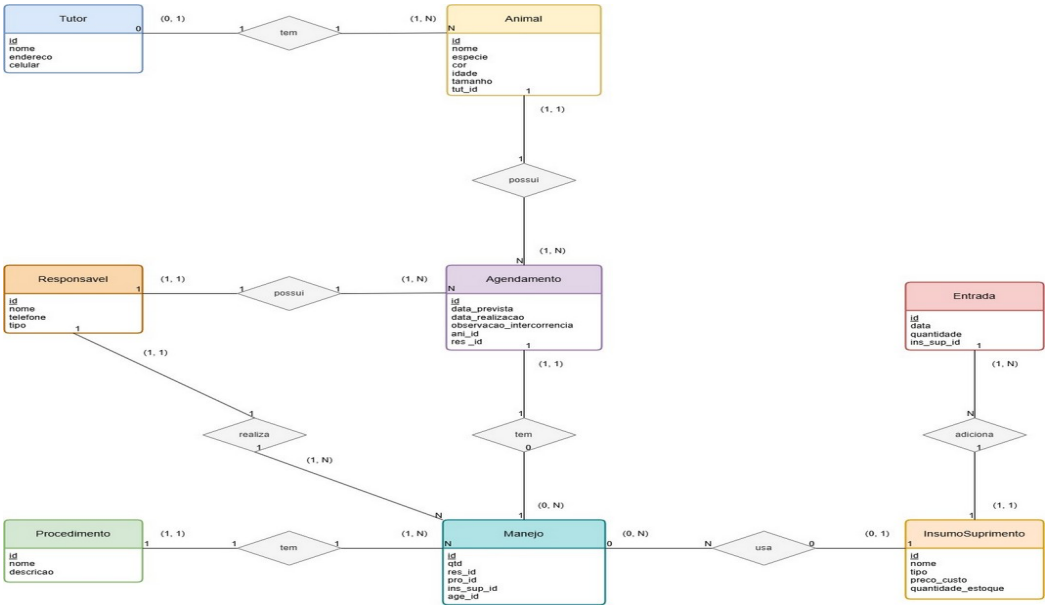
Antes da implementação prática dessas configurações, foi necessário projetar o modelo conceitual do banco de dados, a fim de estruturar corretamente as informações que seriam persistidas pela aplicação.

4.1 DER

O Diagrama Entidade-Relacionamento (DER), visto na **Figura 4.1**, representa graficamente as entidades do sistema e seus relacionamentos. Esse modelo foi elaborado como etapa inicial de planejamento do banco de dados, permitindo identificar as tabelas, atributos, chaves primárias e chaves estrangeiras que compõe o sistema.

No projeto **castracaoAnimal**, o DER auxiliou o mapeamento das entidades Tutores, Animais, Castrações e Vendas, assegurando que a estrutura do banco refletisse corretamente as regras de negócio e mantivesse a consistência dos dados antes da implementação no H2 por meio do *Hibernate*.

Figura 4.1 – Diagrama Entidade-Relacionamento (DER)



Fonte: Elaborado pelo autor (2025).

4.2 H2 Console

O referido banco de dados H2 dispõe de uma interface Word Wide Web (web*)* denominada **H2 Console** (um banco relacional em memória ou arquivo), que permite

visualizar e manipular os dados armazenados nas tabelas do seu projeto. Por meio dessa interface, é possível examinar todas as tabelas criadas pelo *Hibernate* ou definidas manualmente, além de executar comandos *Structured Query Language* (SQL) como *SELECT*, *INSERT*, *UPDATE e DELETE*).

Além disso, o H2 Console facilita a verificação da integridade dos dados, permitindo conformar se as informações estão sendo gravadas corretamente pelo sistema. Essa ferramenta também se mostra bastante útil em etapas de teste e depuração, pois possibilita executar consultas diretamente no banco, sem a necessidade de criar uma interface própria na aplicação.

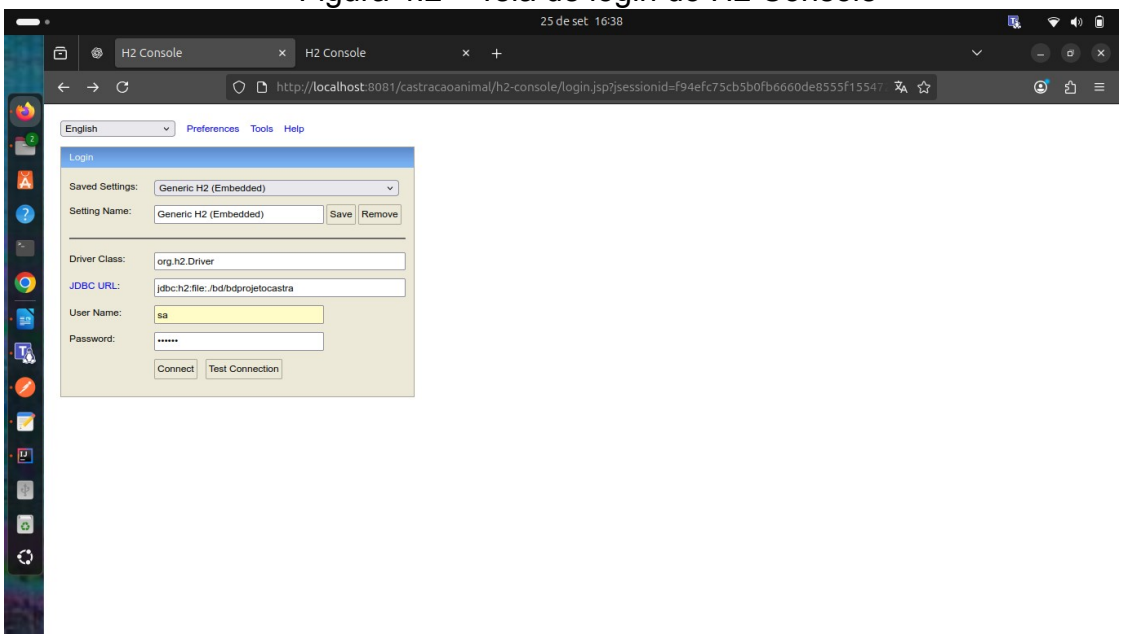
Abaixo segue o link de acesso local do H2 console, seguido da **Figura 4.2** referente ao seu *login* e como acessá-lo.
http://localhost:8081/castracaoanimal/h2-console/login.jsp?jsessionid=fe188a6c127ff54c1bcc5c813255fea2

Na tela de login do H2 Console, fica assim preenchido:

- Driver Class: org.h2.Driver
- JDBC URL: jdbc:h2:file:./bd/bdprojetocastra
- User Name: sa
- Password: 123456

Clica-se em **Connect** para estabelecer a conexão.

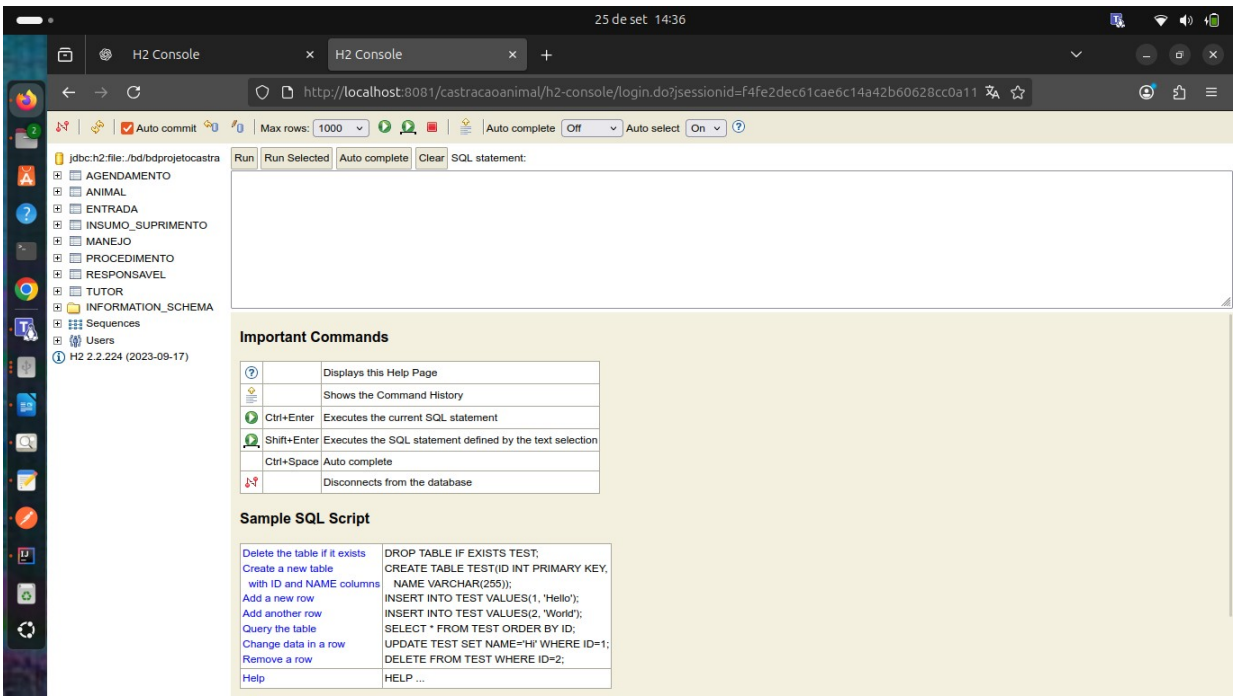
Figura 4.2 – Tela de login do H2 Console



Fonte: Elaborado pelo autor (2025).

Após a configuração inicial da aplicação, torna-se necessário estabelecer a conexão com o banco de dados utilizado. Nesse contexto, foi adotado o banco de dados em memória H2, amplamente utilizado em ambientes de desenvolvimento devido à sua leveza e facilidade de integração com aplicações Spring Boot.

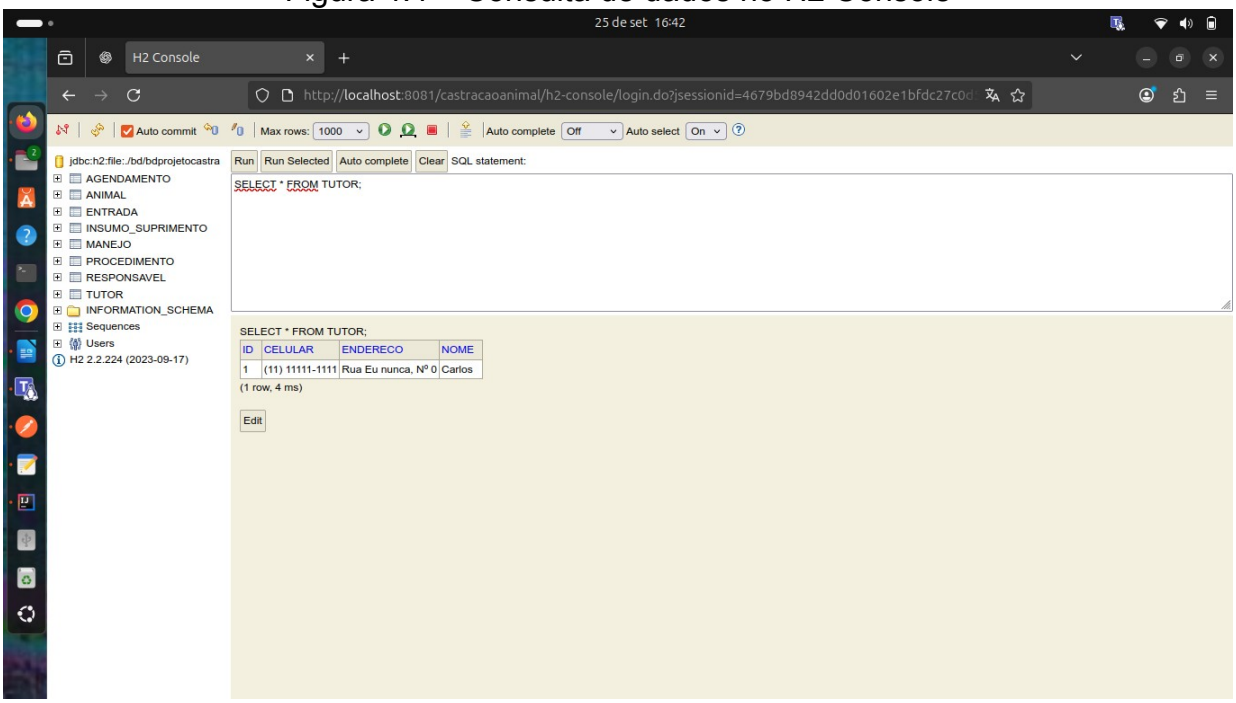
Figura 4.3 – Tela principal do H2 Console



Fonte: Elaborado pelo autor (2025).

A **Figura 4.4** representa como é realizada a consulta ao banco de dados.

Figura 4.4 – Consulta de dados no H2 Console

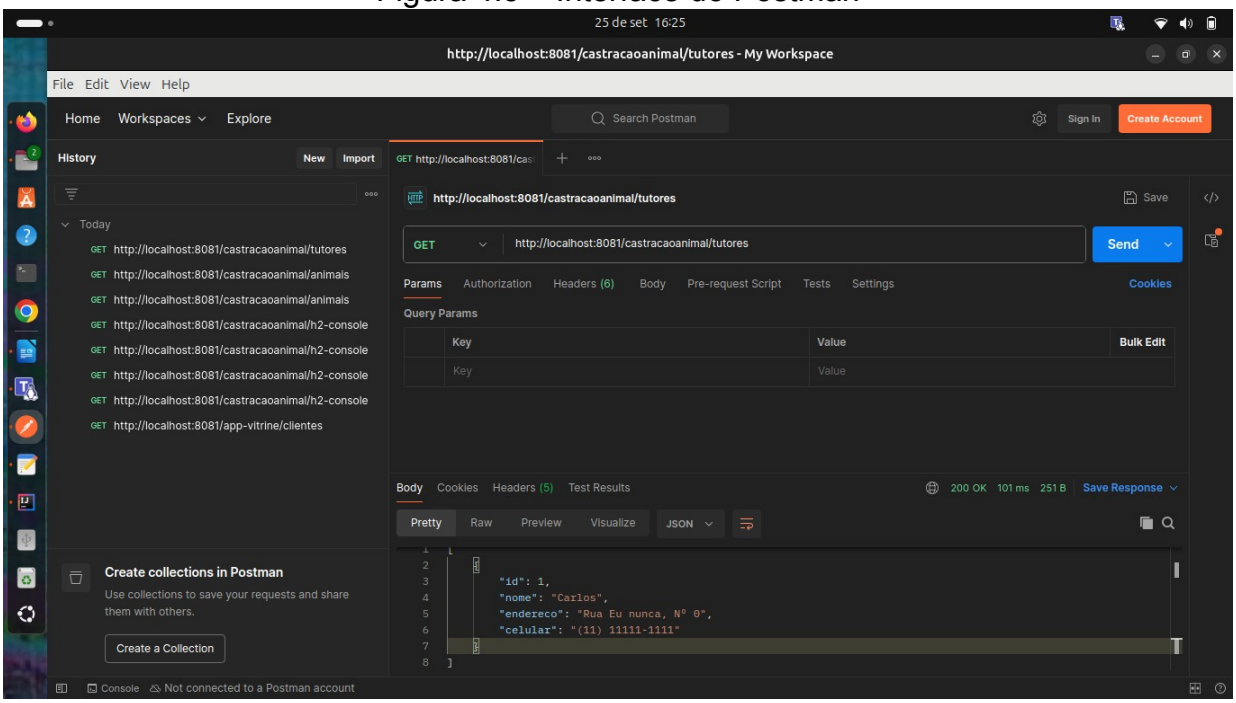


Fonte: Elaborado pelo autor (2025).

4.3 Postman

É uma ferramenta para testar API's, *Representational State Transfer* (REST). Permite enviar requisições *Hypertext Transfer Protocol* (HTTP) (GET, POST, PUT, DELETE) e ver as respostas. Testa se os *endpoints* da API estão funcionando. Permite enviar dados em *JavaScript Objext Notation* (*JSON)* para criar ou atualizar registros. Mostra a resposta da API (*JSON, status code, headers*). Também verifica se a API está funcionando conforme esperado. Ajuda a depurar erros, como 404 (rota não encontrada) ou 500 (erro interno). Fundamental para desenvolvimento e testes, sem precisar criar interfaces de usuário, conforme mostra **Figura 4.5**.

Figura 4.5 – Interface do Postman



Fonte: Elaborado pelo autor (2025).

O uso do *Postman* mostrou-se essencial para validar o correto funcionamento da API desenvolvida, garantindo que cada requisição e resposta ocorresse conforme o esperado. Por meio dessa ferramenta, foi possível testar os e*ndponts* individualmente, identificar falhas de comunicação e assegurar que as operações CRUD estivessem devidamente implementadas. Além disso, o *Postman* contribuiu significativamente para o processo de depuração e refinamento da aplicação, permitindo que ajustes fossem realizados de forma rápida e eficiente antes da disponibilização da API.

4.4 *Endpoints*

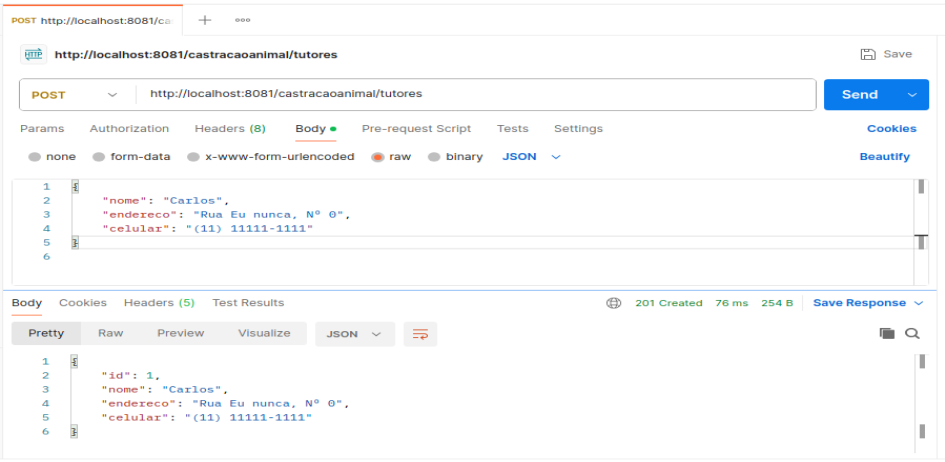
Endpoints são os pontos de acesso de uma API, responsáveis por permitir a comunicação entre o cliente e o servidor. Cada *Endpoints* é uma URL específica que o cliente, com o Postman, utiliza para interagir com o servidor, permitindo que a API receba as requisições (*GET, POST, PUT, DELETE*, etc) e retornando respostas estruturadas, geralmente em JSON, contendo os dados solicitados ou o resultado de alguma ação.

- Tutores

  O endpoint/**tutores** é responsável por cadastrar e gerenciar os tutores dos animais atendidos pela instituição, conforme ilustrado na **Figura 4.6**.

  http://localhost:8081/castracaoanimal/**tutores**

Figura 4.6 – Inserção de dados no endpoint/**tutores**



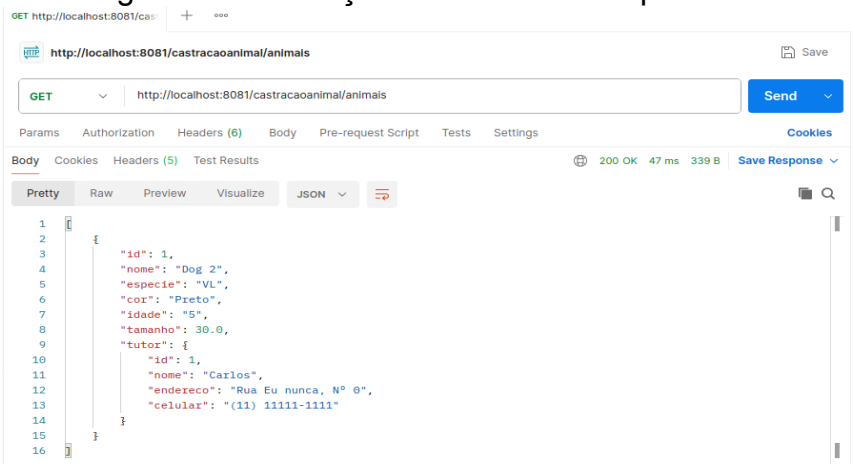
Fonte: Elaborado pelo autor (2025).

- Animais

  O endpoint/**animais** permite o registro e a consulta das informações dos animais sob responsabilidade dos tutores cadastrados, conforme representado na **Figura 4.7.**

  http://localhost:8081/castracaoanimal/**animais**

Figura 4.7 – Inserção de dados no endpoint/**animais**



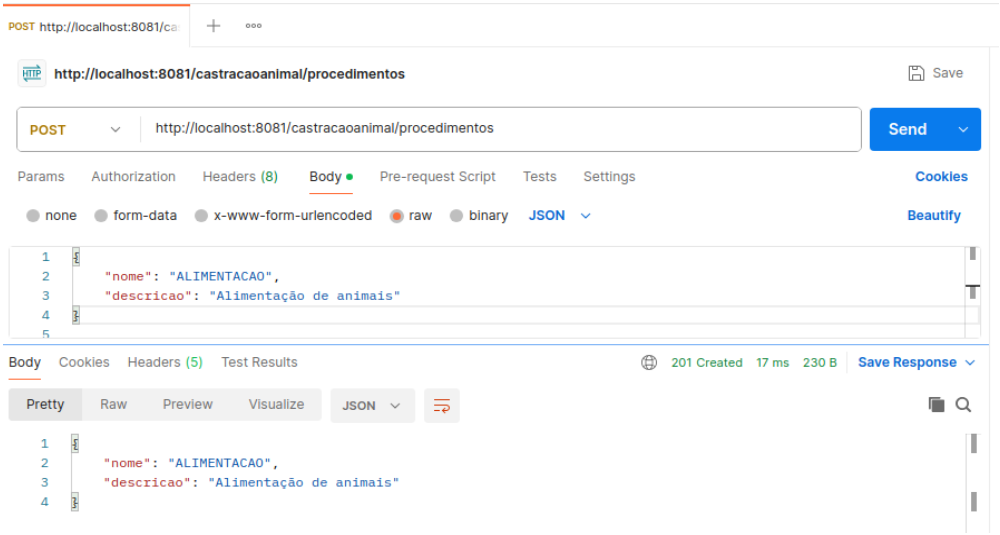
Fonte: Elaborado pelo autor, 2025.

- Procedimentos

  O endpoint/**procedimentos** registra os tipos de procedimentos realizados, como castração, vacinação e atendimento clínico.

  http://localhost:8081/castracaoanimal/**procedimentos**

Figura 4.8 – Inserção de dados no endpoint/**procedimentos**



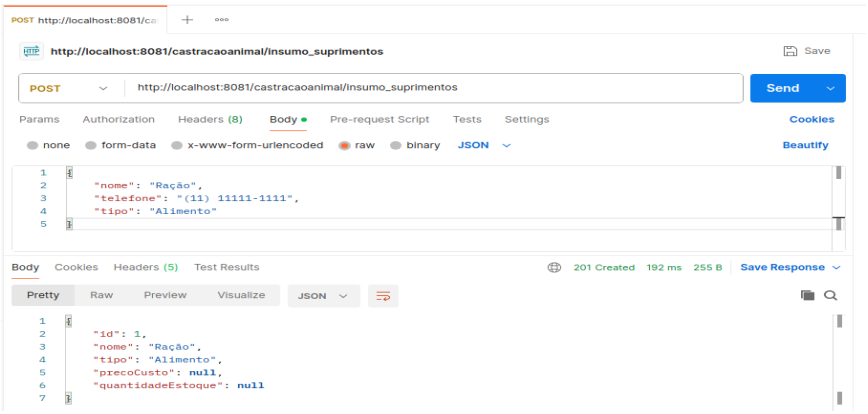
Fonte: Elaborado pelo autor (2025).

- Insumo e Suprimentos

  O endpoint/**insumo_suprimentos** controla os materiais utilizados nas castrações, como medicamentos, seringas e equipamentos.

  http://localhost:8081/castracaoanimal/**insumo_suprimentos**

Figura 4.9 – Inserção de dados no endpoint/**insumo_suprimentos**



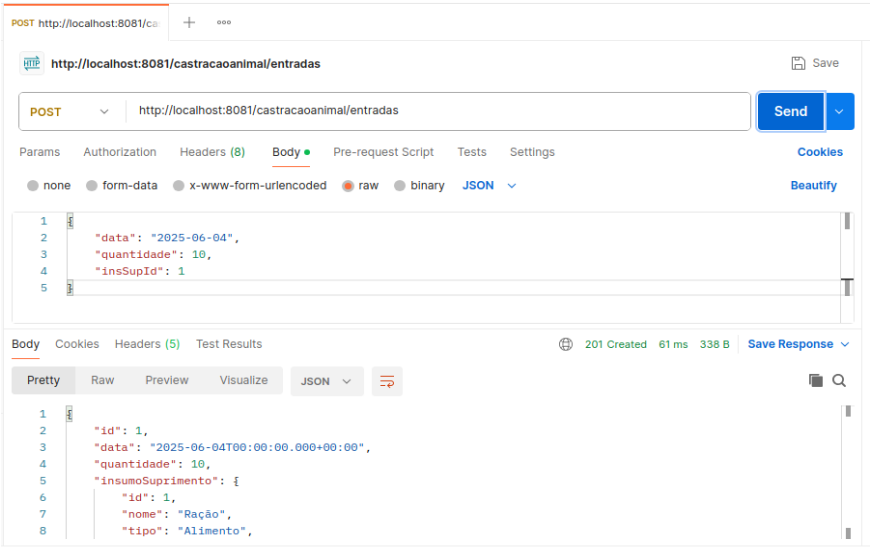
Elaborado pelo autor, 2025.

- Entradas

  O endpoint/**entradas** é utilizado para registrar as entradas de insumos e doações recebidas pela instituição.

  http://localhost:8081/castracaoanimal/**entradas**

Figura 4.10 – Inserção de dados no endpoint/**entrada**



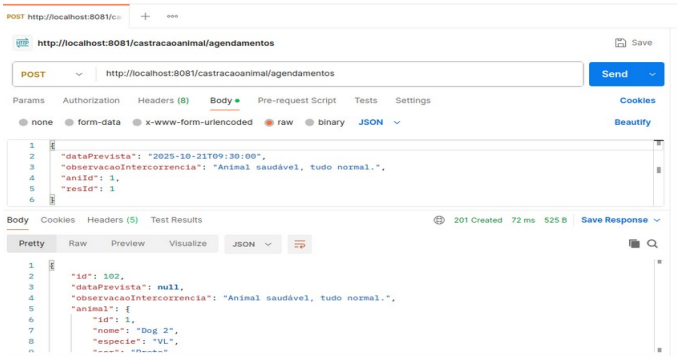
Elaborado pelo autor (2025).

- Agendamentos

  O endpoint/**agendamentos** permite o controle das datas e horários das castrações agendadas, vinculando tutores, animais e procedimentos.

  http://localhost:8081/castracaoanimal/**agendamentos**

Figura 4.11 – Inserção de dados no endpoint/**agendamentos**



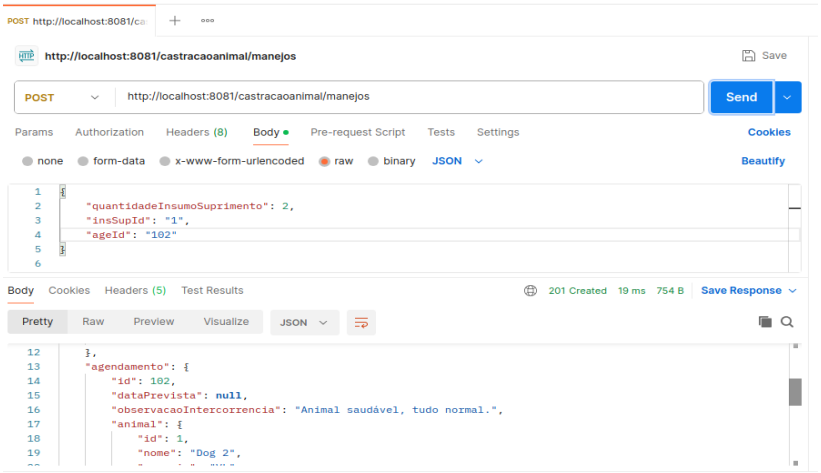
Elaborado pelo autor (2025).

- Manejos

  O endpois/**manejos** é responsável por registrar atividades relacionadas ao cuidado e acompanhamento pós-operatório dos animais

http://localhost:8081/castracaoanimal/**manejos**

Figura 4.12 – Inserção de dados no endpoint/**manejos**



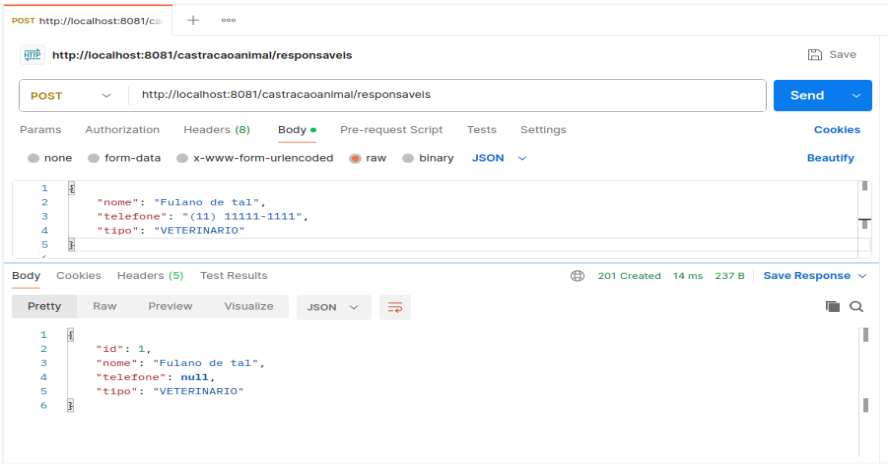
Elaborado pelo autor (2025).

- Responsáveis

  O endpoinr/**responsaveis** armazena os dados dos profissionais e voluntários responsáveis pelos procedimentos e menos realizados.

http://localhost:8081/castracaoanimal/**responsaveis**

Figura 4.13 – Inserção de dados no endpoint/**responsaveis**

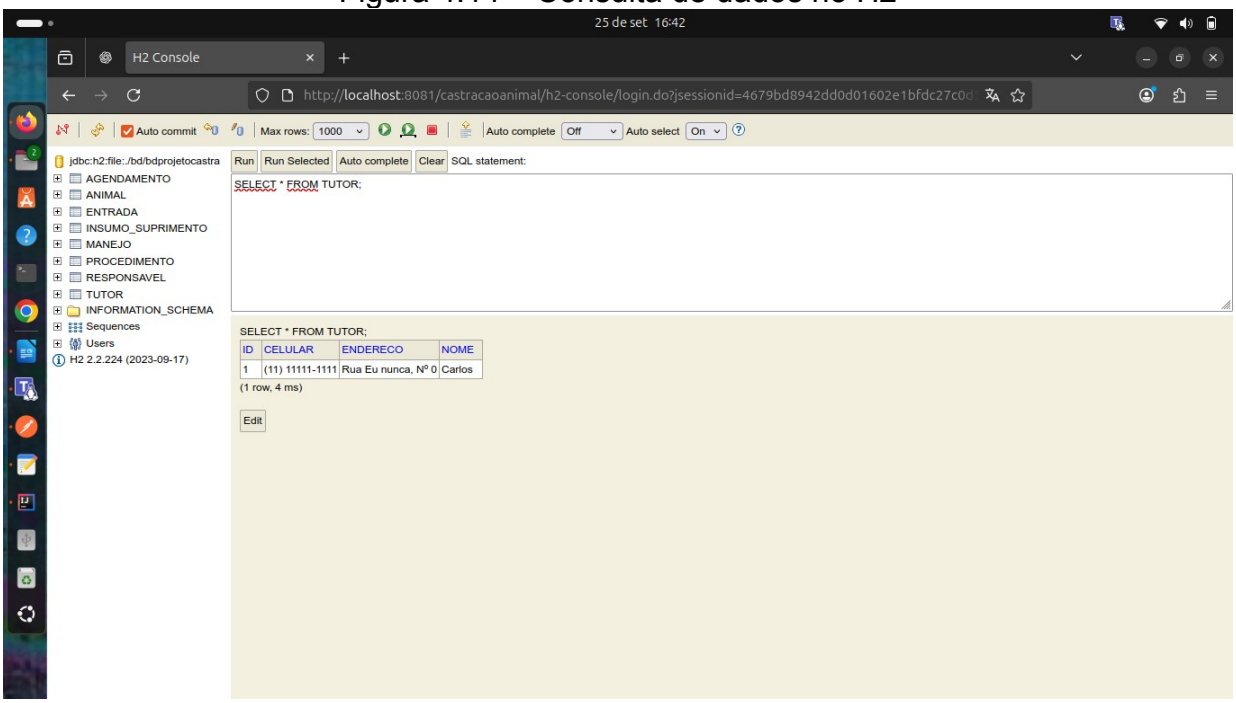


Elaborado pelo autor (2025).

O conjunto de *endpoints* apresentados compõe a estrutura principal da API **“castracaoAnimal”**, permitindo a interação completa com o banco de dados H2 por meio da operação CRUD. A utilização do *Postman* como ferramenta de teste possibilitou validar a funcionalidade de cada rota, assegurando o correto envio e recebimento de dados em formato JSON, além de confirmar a integridade e a consistência das respostas geradas pela aplicação.

Na **Figura 4.14**, observa-se a inserção de dados através de uma porta da API, que permite a comunicação entre o cliente e o servidor. No exemplo mostrado, a requisição realiza a operação `SELECT * FROM TUTOR`, retornando todos os registros da tabela TUTORES.

Figura 4.14 – Consulta de dados no H2



Fonte: Elaborado pelo autor (2025).

Dessa forma, a utilização do console H2 em conjunto com a API possibilitou validar o fluxo completo de persistência e consulta de dados, assegurando que as operações de armazenamento, atualização e recuperação estivessem funcionando corretamente no sistema desenvolvido.

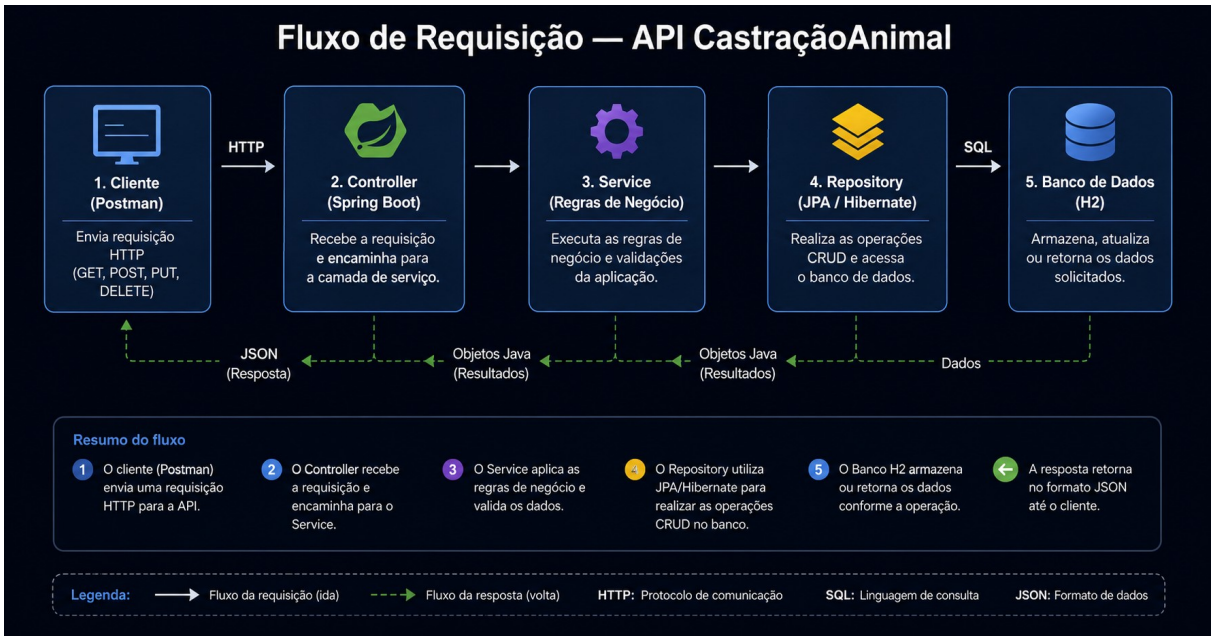
4.5 Hibernate

O Hibernate é um *framework* Java de mapeamento objeto-relacional (ORM), ou seja, atua como uma ponte entre objetos Java (suas classes) e tabelas do banco de dados (como H2, MySQL, etc.). Por meio dele, os objetos Java são automaticamente convertidos em registros no banco (operações *INSERT*), assim como, ao realizar consultas (*SELECT*), esses registros são transformados novamente em objetos Java.

Além disso, o Hibernate é capaz de gerar ou atualizar as tabelas do banco de dados com base nas classes anotadas com *@Entity* e anotações do JPA. Dessa forma, ele facilita a implementação das operações CRUD, eliminando a necessidade de escrever SQL manuais para todas as interações com o banco. Consequentemente, isso reduz erros, permite trabalhar com objetos Java de forma mais natural e mantém o banco de dados sincronizado com as classes do projeto.

Em síntese, o Hibernate é essencial para que a API consiga ler e grava dados no banco de dados H2 automaticamente. De maneira simplificada, pode ser comparado a um "tradutor automático" entre Java e SQL.

Figura 4.15 – Fluxo de requisição da API Castração Animal



Fonte: Elaborado por ChatGPT (2026), sob orientação do autor.

1. O cliente (Postman) envia uma requisição HTTP (GET, POST, PUT ou DELETE).

2. O Controller (Spring Boot) recebe a requisição e direciona para o serviço apropriado.

3. O Service executa as regras de negócio e validações necessárias.

4. O Repository, utilizando Hibernate/JPA, traduz os objetos Java em instruções SQL.

5. O banco de dados H2 armazena, atualiza ou retorna os dados solicitados.

6. A resposta é enviada de volta ao cliente no formato JSON.

O prompt da imagem gerada por IA:

Crie uma imagem ilustrando o fluxo de requisição da API CastraçãoAnimal, representando visualmente a comunicação entre os componentes da aplicação.

- Utilize um estilo técnico e moderno, com fundo escuro e caixas em tons de azul.
- O diagrama deve conter os seguintes elementos conectados por setas:
- Cliente (*Postman*) → enviando uma requisição HTTP para o *Controller* (*Spring Boot*);
- O *Controller* realiza operações CRUD via JPA com o *Repository* (*Hibernate*/JPA);
- O *Repository* se comunica com o Banco H2, responsável pelo armazenamento dos dados;
- O retorno ocorre no sentido inverso (Banco → *Repository* → *Controller* → Cliente) em formato JSON.
- Inclua o título “Fluxo de Requisição — API CastraçãoAnimal” centralizado no topo da imagem.

4.6 Estrutura de pastas e funções das classes

4.6.1 Controle (*Controller*)

Exemplo: *perfilController.class, produtoController.class, VitrineController.class*.

Função: Receber e tratar as requisições HTTP do cliente (como POST, GET, PUT, DELETE) e direcioná-las para os serviços ou repositórios correspondentes.

Cada *controller* corresponde a uma entidade ou funcionalidade principal da aplicação:

PerfilController: gerencia requisições relacionadas a perfis de usuários ou clientes.

ProdutoController: gerencia requisições sobre produtos disponíveis na loja.

VitrineController: gerencia requisições relacionadas à vitrine (exibição de produtos).

4.6.2 DTO (*Data Transfer Object*)

Exemplo: perfilDTO.class, produtoDTO.class, vitrineDTO.class

Função: Servir como objeto de transporte de dados entre camadas da aplicação, geralmente do *controller* para a camada de serviço ou vice-versa.

Permite selecionar apenas os dados relevantes que serão enviados ou recebidos, evitando expor informações sensíveis ou desnecessárias.

4.6.3 Modelo (Model ou Entidade)

Exemplo: perfil.class, produto.class, vitrine.class

Função: Representar a estrutura de dados da aplicação, geralmente mapeando tabelas do banco de dados.

Cada classe modelo corresponde a uma tabela ou conceito de negócio:

Perfil: dados do usuário ou cliente.

Produto: informações sobre produtos da loja.

Vitrine: informações sobre produtos exibidos na vitrine.

4.6.4 Repositório (*Repository*)

Exemplo: perfilRepository.class, produtoRepository.class, vitrineRepository.class

Função: Responsável pelo acesso e manipulação dos dados no banco.

O *Spring Data JPA* fornece métodos prontos (como save(), findById(), delete()) e permite criar consultas personalizadas.

Cada repositório está associado a um modelo específico, garantindo que cada entidade tenha suas operações de persistência centralizadas.

Em resumo:

- ***Model***: representa os dados (Modelo/Entidade).
- ***Controller***: recebe e responde às requisições do usuário.
- **DTO**: transporta dados entre camadas de forma segura.
- ***Repository***: faz a conexão entre a aplicação e o banco de dados.

Com o desenvolvimento da API Castração Animal, foi possível integrar diversas tecnologias modernas, como *Spring Boot*, H2 e *Hibernate*, em um projeto com relevância social e aplicabilidade real. O sistema permite gerenciar tutores, animais e procedimentos de forma ágil e segura, demonstrando o potencial da tecnologia em apoiar iniciativas públicas de bem-estar animal.

4.7 Considerações Técnicas

O desenvolvimento da API CastraçãoAnimal proporcionou uma experiência prática de integração entre diversas tecnologias e conceitos da programação *web* moderna. Por meio da utilização do Spring Boot, Hibernate/JPA e do banco H2, foi possível compreender de forma aplicada o funcionamento de uma arquitetura em camadas e processo de persistência de dados em sistemas Java.

Além disso, o uso de ferramentas como o Postman e o H2 Console permitiu testar, validar e depurar o fluxo completo de comunicação entre cliente e servidor, consolidando o aprendizado sobre operações CRUD, requisições HTTPS e formato JSON.

De maneira geral, o capítulo demonstrou como a aplicação dos conhecimentos teóricos pode resultar em soluções tecnológicas com impacto social, evidenciando o papel da programação no apoio a iniciativa pública e no fortalecimento do bem estar animal.