UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR

DAVI CARDOSO DE OLIVEIRA
RAFAEL SOUZA DE ALMEIDA

# SIGMEV - Sistema de Gerenciamento de Medicamentos Veterinários

Prof. Filipe Braida do Carmo, D.Sc.
Orientador

Nova Iguaçu, Dezembro de 2025

# SIGMEV - Sistema de Gerenciamento de Medicamentos Veterinários

**Davi Cardoso de Oliveira**

**Rafael Souza de Almeida**

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____

Davi Cardoso de Oliveira

_____

Rafael Souza de Almeida

Aprovado por:

_____

Prof. Filipe Braida do Carmo, D.Sc.

_____

Prof. Viviane de Souza Magalhães, D.Sc.

_____

Prof. Natália Chaves Lessa, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Dezembro de 2025

***DOCUMENTOS COMPROBATÓRIOS Nº 33050/2025 - CoordCGCC (12.28.01.00.00.98)***

***(Nº do Protocolo: NÃO PROTOCOLADO)***

***(Assinado digitalmente em 12/12/2025 08:36 )***
*FILIPE BRAIDA DO CARMO*
*PROFESSOR DO MAGISTERIO SUPERIOR*
*DeptCC/IM (12.28.01.00.00.83)*
*Matrícula: ###295#4*

***(Assinado digitalmente em 12/12/2025 13:33 )***
*NATALIA CHAVES LESSA*
*PROFESSOR DO MAGISTERIO SUPERIOR*
*DeptCC/IM (12.28.01.00.00.83)*
*Matrícula: ###435#4*

***(Assinado digitalmente em 11/12/2025 15:11 )***
*VIVIANE DE SOUZA MAGALHAES*
*FARMACEUTICO-HABILITACAO*
*DCFar (12.28.01.00.00.00.47)*
*Matrícula: ###463#5*

***(Assinado digitalmente em 11/12/2025 15:12 )***
*DAVI CARDOSO DE OLIVEIRA*
*DISCENTE*
*Matrícula: 2022######8*

***(Assinado digitalmente em 11/12/2025 14:34 )***
*RAFAEL SOUZA DE ALMEIDA*
*DISCENTE*
*Matrícula: 2022######9*

*Visualize o documento original em https://sipac.ufrrj.br/documentos/ informando seu número: **33050**, ano: **2025**, tipo: **DOCUMENTOS COMPROBATÓRIOS**, data de emissão: **11/12/2025** e o código de verificação: **c4df45aa71***

# Agradecimentos

Davi Cardoso de Oliveira

Em primeiro lugar, agradeço a Deus, pela vida, pelas oportunidades, pela sabedoria concedida e por me sustentar nos momentos difíceis e de incerteza. Sua presença foi essencial em cada etapa desta caminhada acadêmica.“Até os jovens se cansam e ficam exaustos, e os moços certamente tropeçam, mas aqueles que esperam no SENHOR renovam as suas forças” (Isaías 40:30-31).

Em especial, agradeço aos meus pais, Leandro e Danielle, pelo amor, conselhos, paciência e por todo o apoio concedido durante todos os anos; sempre terão a minha gratidão. Agradeço também aos meus irmãos e familiares, que sempre estiveram ao meu lado, oferecendo força e apoio incondicional.

Aos meus amigos da graduação, que dividiram comigo não apenas os desafios do curso, mas também os bons momentos; a boa companhia deles nas risadas, nos trabalhos em grupo e nas longas horas de estudo foram especiais. A amizade de vocês tornou essa jornada mais leve e memorável, sem vocês os dias na universidade seriam somente "mais um dia", porém vocês ajudaram a tornar cada dia como único.

Aos professores da Universidade Federal Rural do Rio de Janeiro, que ao longo dos anos contribuíram de forma decisiva para minha formação acadêmica e pessoal. Um agradecimento especial ao meu orientador, Prof. Filipe Braida do Carmo, por toda a paciência, dedicação e apoio ao longo do desenvolvimento deste trabalho. Sua orientação técnica e seus conselhos foram cruciais. Obrigado por acreditar neste projeto e me incentivar a seguir sempre em frente.

Rafael Souza de Almeida

Agradeço ao meu Senhor e Salvador Jesus Cristo; sem Sua força neste momento, jamais teria conseguido."Ele é a imagem do Deus invisível, o primogênito de toda a criação. Pois nele foram criadas todas as coisas, nos céus e sobre a terra, as visíveis e as invisíveis, sejam tronos, sejam soberanias, quer principados, quer potestades. Tudo foi criado por meio dele e para ele." (Colossenses 1:15-16).

Agradeço aos meus pais, Fábio e Adriana, e também ao meu irmão, Gustavo. Todo o apoio deles foi crucial durante esse período, e agradeço a Deus por terem sido a minha base durante momentos tão difíceis e conturbados. Vocês são um presente de Deus em minha vida. Amo muito vocês.

Um agradecimento especial ao meu orientador, Prof. Filipe Braida do Carmo, pelo acompanhamento próximo e pelas contribuições valiosas ao longo do projeto. Sua orientação e paciência foram decisivas para o sucesso deste trabalho.

Agradeço aos meus amigos de graduação, Davi Cardoso, Maxwell William e Leonardo Aizhu. Passamos por muitas coisas e nem acredito que está finalmente acabando. Agradeço pela amizade de vocês durante todo esse tempo, os momentos felizes e os momentos conturbados. As incontáveis provas que fizemos e as mais variadas discussões sobre diversos assuntos... Levarei vocês em meu coração.

Agradeço aos amigos da minha igreja, em especial ao GAJ, grupo que tanto amo. Posso dizer que não possuo apenas amigos, mas irmãos em Cristo. Cada oração, cada risada e cada semana em que eu contava as coisas que estava passando, vocês estavam lá me ajudando. Sou muito grato a Deus por isso.

Por fim, agradeço a todos que, de alguma forma, deixaram sua marca nessa trajetória. Muito obrigado por fazerem parte dessa conquista.

SDG!

RESUMO

SIGMEV - Sistema de Gerenciamento de Medicamentos Veterinários

Davi Cardoso de Oliveira e Rafael Souza de Almeida

Dezembro/2025

Orientador: Filipe Braida do Carmo, D.Sc.

O uso de medicamentos veterinários é essencial para a saúde animal e para a segurança dos produtos de origem animal. No entanto, sistemas existentes para consulta a esses fármacos possuem limitações recorrentes no acesso a dados confiáveis, atualizados e estruturados sobre medicamentos, frequentemente dispersos em fontes diversas e carentes de padronização. Diante dessa problemática o presente trabalho descreve o desenvolvimento do Sistema de Gerenciamento de Medicamentos Veterinários (SIGMEV), uma plataforma web criada para centralizar, organizar e facilitar o acesso a informações técnicas sobre fármacos utilizados na medicina veterinária. A solução final permite realizar consultas detalhadas, filtragens rápidas e visualização clara de informações essenciais, como princípios ativos, classes farmacológicas, indicações de uso, restrições legais e exigências de prescrição. O trabalho demonstra a relevância do uso de tecnologias modernas no desenvolvimento de sistemas aplicados à saúde animal e sugere caminhos para aprimoramentos futuros, como expansão funcional e integração com fontes externas de dados.

**Palavras-chave**: SIGMEV; Medicamentos veterinários; Sistemas de informação; Consulta farmacológica; Saúde animal.

ABSTRACT

SIGMEV - Sistema de Gerenciamento de Medicamentos Veterinários

Davi Cardoso de Oliveira and Rafael Souza de Almeida

Dezembro/2025

Advisor: Filipe Braida do Carmo, D.Sc.

*The use of veterinary medicines is essential for animal health and for ensuring the safety of products of animal origin. However, existing systems for consulting information on these drugs often present recurring limitations, including the lack of reliable, updated, and structured data, which is frequently dispersed across multiple sources and lacks standardization. In response to this issue, this work presents the development of SIGMEV, a web-based platform designed to centralize, organize, and facilitate access to technical information on pharmaceuticals used in veterinary medicine. The proposed solution enables detailed queries, fast filtering, and clear visualization of essential data such as active ingredients, pharmacological classes, indications, legal restrictions, and prescription requirements. This study highlights the relevance of modern web technologies in the development of information systems applied to animal health and suggests potential directions for future improvements, including functional expansion and integration with external data sources.*

***Keywords***: *SIGMEV; Veterinary medicines; Information systems; Pharmacological consultation; Animal health.*

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviaturas e Siglas

| | |
|---|---|
| **UFRRJ** | Universidade Federal Rural do Rio de Janeiro |
| **PK** | Chave Primária |
| **FK** | Chave Estrangeira |
| **JSX** | *Javascript XML* |
| **SIGMEV** | Sistema de Gerenciamento de Medicamentos Veterinários |
| **DOM** | *Document Object Model* |
| **API** | *Application Programming Interface* |
| **HTML** | *HyperText Markup Language* |
| **CSS** | *Cascading Style Sheets* |
| **URM** | Uso Racional de Medicamentos |
| **FPS** | *Frames Per Second* |
| **UI** | *User Interface* |
| **UX** | *User Experience* |
| **RF** | Requisitos Funcionais |
| **RNF** | Requisitos Não Funcionais |
| **RN** | Regras de Negócio |
| **CU** | Casos de Uso |
| **DER** | Diagrama Entidade Relacionamento |
| **SPA** | *Single Page Application* |
| **MVC** | *Model-View-Controller* |

**ORM** *Object-Relational Mapping*

**SQL** *Structured Query Language*

**CLI** *Command-Line Interface*

**API REST** *Representational State Transfer Application Programming Interface*

**HTTP** *Hypertext Transfer Protocol*

**MCP** *Model Context Protocol*

# Sumário

# Capítulo 1

# Introdução

O uso inadequado ou indiscriminado de medicamentos pode resultar na presença de resíduos químicos em alimentos, no desenvolvimento de resistência antimicrobiana e em impactos negativos à saúde ambiental (Agência Nacional de Vigilância Sanitária (ANVISA), 2019). Na medicina humana, protocolos rígidos de gestão de medicamentos já são consolidados para minimizar erros de prescrição e administração (ANACLETO et al., 2010). Em contrapartida, a medicina veterinária opera em um contexto de maior complexidade biológica e informacional, como apontam Santamaria e Zimmerman (2011): o médico-veterinário precisa lidar com múltiplas espécies e terminologias não padronizadas, o que dificulta a uniformização de dados e procedimentos.

Essa complexidade é agravada pela fragmentação tecnológica, pois segundo Lustgarten et al. (2020), a ausência de integração entre sistemas veterinários cria "silos" de informação, dificultando o acesso a dados clínicos e farmacológicos essenciais. Embora o Ministério da Agricultura, Pecuária e Abastecimento (2021) destaque avanços normativos relacionados ao controle e registro de produtos veterinários, profissionais, pesquisadores e estudantes ainda enfrentam dificuldades para acessar informações técnico-científicas atualizadas e confiáveis (GAMA, 2024).

Além disso, a prática veterinária depende de dados precisos sobre indicações terapêuticas, princípios ativos, dosagens, contraindicações, espécies-alvo e restrições

legais. Segundo Vlasiou (2024), a falta de padronização e integração entre instituições reduz a confiabilidade das informações e aumenta a carga computacional para a integração de dados.

Diante desse cenário, o acesso a informações confiáveis torna-se condição essencial para garantir a segurança terapêutica e promover o Uso Racional de Medicamentos (URM), conceito igualmente fundamental na saúde pública, conforme o Ministério da Saúde (2024). A carência de bases unificadas na veterinária compromete o URM e está associada ao aumento de erros de medicação, situados entre os eventos adversos mais frequentes na rotina clínica (PINHO; NASR-ESFAHANI; PANG, 2024).

Assim, reforça-se a necessidade de sistemas que centralizem, organizem e facilitem o acesso a dados confiáveis. Nesse contexto, a informatização surge como uma estratégia fundamental para otimizar a gestão de informações e apoiar a prática clínica e acadêmica. Ferramentas digitais com interfaces intuitivas, mecanismos de busca eficientes e atualizações contínuas tornam-se indispensáveis para atender às demandas regulatórias e operacionais, especialmente diante das lacunas persistentes na padronização e disponibilização de informações sobre medicamentos veterinários no cenário brasileiro (PAESE; JESUS; ANDRADE, 2023).

## 1.1 Objetivo

Este trabalho propõe o desenvolvimento de um sistema web voltado ao cadastro, gerenciamento e consulta estruturada e detalhada de medicamentos de uso veterinário. Este projeto foi idealizado para ser utilizado por um grupo de extensão vinculado ao Departamento de Medicina Veterinária da Universidade Federal Rural do Rio de Janeiro (UFRRJ). Nesse contexto, o sistema contempla a participação de diferentes perfis de usuários, incluindo alunos, responsáveis pelo cadastro e atualização das informações dos medicamentos; profissionais da saúde, que podem colaborar com a validação, cadastro, revisão e complementação dos dados; e administradores, encarregados do gerenciamento geral da plataforma, do controle de acessos e da manutenção da integridade das informações. Essa abordagem colaborativa busca

integrar ensino, extensão e prática profissional.

O sistema deve assegurar a segurança, a integridade e a manutenção contínua dos dados, atendendo às demandas operacionais dos profissionais da área. Assim, o projeto busca oferecer um ambiente seguro e confiável, no qual especialistas possam supervisionar, inserir, editar e remover informações da base de dados, garantindo a veracidade e a constante atualização do conteúdo disponibilizado. Além disso, propõe-se o desenvolvimento de uma interface responsiva e intuitiva, capaz de se adaptar a diferentes dispositivos e de facilitar a navegação do usuário, promovendo uma experiência de uso acessível, eficiente e consistente.

## 1.2 Organização do Trabalho

- Capítulo 2: Nesse capítulo será abordado a fundamentação teórica do *React* e como seu ecossistema foi a solução para garantia de usabilidade e acessibilidade em diferentes dispositivos, com uma interface responsiva e interativa.

- Capítulo 3: Este capítulo detalha a concepção e o desenvolvimento do sistema web para dados farmacológicos veterinários. São abordados a modelagem de dados, a arquitetura e as funcionalidades, como cadastro, gerenciamento e busca estruturada de medicamentos.

- Capítulo 4: Descreve todas as tecnologias utilizadas no desenvolvimento, a estrutura e arquitetura do projeto, além de mostrar as interfaces do mesmo para o usuário e o administrador.

- Capítulo 5: Resume os principais resultados do trabalho, discute as limitações encontradas e propõe sugestões para futuras pesquisas e aprimoramentos do sistema.

# Capítulo 2

# React

Diante da necessidade de um sistema web moderno, intuitivo e capaz de organizar informações de forma eficiente, torna-se fundamental selecionar tecnologias que ofereçam estabilidade, ampla documentação e um conjunto de ferramentas robusto. Para o desenvolvimento da interface escolhemos a biblioteca *React* por possuir um ecossistema maduro e abrangente, além de apresentar a maior popularidade entre desenvolvedores, conforme pesquisa realizada em 2023[1].

Esse ecossistema torna o *React* especialmente adequado para a construção de interfaces dinâmicas, escaláveis e de fácil manutenção, alinhando-se diretamente às demandas do sistema proposto. Esta escolha justifica-se pela sua ampla adoção no desenvolvimento web moderno e pela capacidade de proporcionar experiências interativas, rápidas e dinâmicas aos usuários. Neste capítulo, abordaremos sobre a definição desta principal tecnologia utilizada neste projeto, seu contexto histórico onde foi pensada e criada, sua fundamentação teórica e como a utilizamos neste trabalho.

[1] <https://survey.stackoverflow.co/2023/>

## 2.1 Definição

Segundo sua própria documentação[2], o *React* é uma biblioteca *JavaScript* de código aberto desenvolvida pela *Meta* (*Facebook*), voltada para a construção de interfaces de usuário a partir de componentes reutilizáveis, o que facilita a modularização e a manutenção de aplicações web modernas (BANKS; PORCELLO, 2020). A ideia fundamental do *React* é que qualquer interface, independentemente de sua complexidade, pode ser dividida em pequenos blocos de construção independentes chamados componentes. Um componente encapsula sua própria lógica, sua própria marcação por meio de uma sintaxe chamada *Javascript XML* (JSX) e, opcionalmente, seu próprio estado interno (GACKENHEIMER, 2015). Com base nessa ideia, em uma aplicação web, a barra de busca pode ser um componente, a lista de resultados pode ser outro, e cada item individual dessa lista pode ser um componente. Esses componentes menores podem, então, ser combinados, dando origem a componentes maiores e, por fim, à construção de toda a página.

Outro aspecto essencial, que também é definido na documentação, é a utilização do *Virtual Document Object Model* (DOM). O DOM virtual é definido como um conceito de programação em que uma representação ideal, ou virtual, de uma interface do usuário é mantida na memória e sincronizada com o DOM real por uma biblioteca como o *ReactDOM*[3]. O desenvolvedor não interage diretamente com a *Application Programming Interface* (API) do DOM como no *Javascript*; em vez disso, fornece instruções que ele deseja que o *React* construa. Este, por sua vez cuidará da renderização e reconciliação dos elementos. Isso possibilita um processo de atualização eficiente ao aplicar no DOM real apenas as modificações necessárias, em um processo otimizado que resulta em uma aplicação rápida e fluida (FEDOSEJEV, 2015).

[2] <https://pt-br.legacy.reactjs.org/>
[3] <https://legacy.reactjs.org/docs/faq-internals.html>

## 2.2 Contexto Histórico

O *JavaScript*, surgiu em 1995, criado por Brendan Eich na *Netscape Communications*, com o objetivo inicial de permitir maior interatividade em páginas web (FLANAGAN, 2020). Nos primeiros anos, a linguagem era utilizada apenas para tarefas simples, como validação de formulários e pequenas animações, sendo vista como uma ferramenta de apoio ao *HyperText Markup Language* (HTML) e *Cascading Style Sheets* (CSS). Com o passar do tempo, sua evolução foi marcada pela padronização do *ECMAScript*, que estabeleceu uma base comum para implementação nos navegadores (ZAKAS, 2016).

Essa evolução consolidou o *JavaScript* não apenas como uma linguagem de *script*, mas como o núcleo da web moderna, tornando-se um dos pilares essenciais no desenvolvimento de aplicações interativas e dinâmicas. Conforme destacam FLANAGAN (2020) e ZAKAS (2016), o *JavaScript* deixou de ser uma ferramenta auxiliar para assumir um papel central na construção de experiências web ricas e responsivas, impulsionado pela padronização do *ECMAScript*, a popularização de *frameworks* e bibliotecas modernas.

No entanto, o crescimento da complexidade das aplicações trouxe limitações relevantes na manipulação direta do DOM e na organização do código. Desenvolvedores passaram a enfrentar problemas de performance, duplicação de lógica e dificuldade de manutenção conforme o tamanho das aplicações aumentava (BANKS; PORCELLO, 2020). Foi nesse contexto que começaram a surgir soluções voltadas para a componentização e para um melhor gerenciamento de estado, como o *AngularJS* e o *Backbone.js*, cada uma propondo diferentes abordagens para o desenvolvimento web modular e reativo (POPA; ALBERT, 2018). A necessidade de uma arquitetura mais eficiente e previsível motivou o desenvolvimento de novas ferramentas e, entre elas, o *React*, que viria a redefinir a maneira como as interfaces são construídas.

O *React* surgiu em 2011 como uma solução interna desenvolvida pela equipe de engenharia do *Facebook*, liderada por Jordan Walke, com o propósito de otimizar o desempenho e a manutenção das interfaces da plataforma, que enfrentava desa-

fios de escalabilidade e lentidão nas atualizações dinâmicas de conteúdo (BANKS; PORCELLO, 2020). A primeira implementação pública ocorreu em 2013, durante a conferência *JSConf US*, marcando o início de uma nova era no desenvolvimento *front-end* ao introduzir o conceito de *component-based architecture* e a ideia do *Virtual*DOM, que revolucionaram a forma como aplicações web reativas passaram a ser concebidas (ACCOMAZZO; MURRAY; LERNER, 2017).

A filosofia por trás do *React* prioriza a previsibilidade e a simplicidade no fluxo de dados, sintetizada pelo princípio do *one-way data binding*, no qual a informação flui em uma única direção, dos componentes pais para os filhos, reduzindo a complexidade de estados e interações (BANKS; PORCELLO, 2020). Com o passar dos anos, o *React* passou a incorporar novos recursos e paradigmas, como o uso de *hooks*, introduzidos em 2019, que permitiram a utilização de estado e ciclo de vida em componentes funcionais [4].

### 2.2.1 Atualizações

A arquitetura *Fiber*, introduzida na versão 16, ampliou ainda mais a capacidade do *React* de lidar com atualizações concorrentes e complexas (CLARK, 2017). O *Fiber* divide as tarefas de renderização em unidades menores, chamadas *fibers*, que podem ser pausadas, priorizadas e retomadas, permitindo que interações do usuário, como cliques ou rolagem, sejam respondidas imediatamente, mesmo enquanto grandes atualizações de interface estão em andamento (ABRAMOV, 2019). Essa abordagem cooperativa aproxima o *React* de um modelo de execução concorrente, aumentando a responsividade da aplicação sem exigir mudanças drásticas na lógica do programador.

Na versão 16.8 foram introduzidos os *Hooks*, que possuem o papel de permitir o uso de estado e outros recursos do *React* sem a necessidade da criação de uma classe. Também se tornou possível criar *Hooks* próprios para compartilhar lógica *stateful* reutilizável entre os componentes. Isso tornou o *React* ainda mais flexível, pois em código complexo, a simplicidade dos componentes funcionais com *Hooks* contribui para a legibilidade e depuração do que o uso de *this* e o ciclo de vida

[4] <https://react.dev/learn>

complexo das classes, o que leva a uma redução significativa de *bugs* (ABRAMOV, 2019). Isso não apenas simplifica a composição de componentes, mas também facilita a reutilização de lógica de estado entre diferentes partes da aplicação, promovendo maior modularidade e previsibilidade.

Segundo DODDS (2018), os *Hooks* permitem refatoração e reúso seguros por meio da criação de *Custom Hooks*, permitindo que desenvolvedores extraiam a lógica com estado para uma função reutilizável. Isso reduz o potencial de erros e promove a segurança, especialmente em bases de código extensas, pois a lógica complexa é isolada e pode ser testada. Encerrando a versão 16 na atualização 16.13.0, o foco esteve apenas na correção de *bugs* e na descontinuação de alguns aspectos, tendo em vista que, por ser a última atualização da versão 16, a equipe de desenvolvedores já estava se preparando para a nova versão que viria a ser lançada.

Na versão 17, o *React* não buscou grandes atualizações de API; em vez disso, o foco foi facilitar seu uso. Em particular, o *React* 17 é uma versão de transição que torna mais seguro incorporar uma árvore gerenciada por uma versão do *React* dentro de uma árvore gerenciada por outra versão (ABRAMOV; NABORS, 2020). Nessa versão, o *React* ainda funcionava essencialmente como um renderizador síncrono, no qual a maioria das atualizações era tratada de forma linear; isto é, uma vez iniciado o processo de renderização, o *React* não podia interrompê-lo, reordená-lo ou atribuir prioridades diferentes a partes da interface (KAPOOR, 2022).

Era até eficiente para aplicações menores, mas começou a se tornar um gargalo conforme os projetos cresceram, pois qualquer atualização iniciava um ciclo de renderização que precisava ser completado antes que o navegador pudesse responder a novas interações do usuário, então, em casos de componentes mais pesados ou listas extensas, isso causava travamentos momentâneos, quedas de *Frames Per Second* (FPS) e sensação de lentidão [5].

A versão 18 introduziu uma série de mecanismos projetados especificamente para superar essas limitações e gargalos das versões anteriores, buscando tornar o ecossistema mais preparado para cenários concorrentes. Uma das principais melhorias foi a

[5] <https://legacy.reactjs.org/blog/2022/03/29/react-v18.html>

introdução do *automatic batching*, que agrupa automaticamente diversas atualizações de estado, inclusive fora do escopo de eventos *React*, reduzindo renderizações desnecessárias e melhorando a performance da aplicação [6]. Adicionalmente, APIs como *startTransition* e *useDeferredValue* permitem classificar atualizações como urgentes ou não-urgentes, conferindo ao desenvolvedor controle fino sobre o agendamento das renderizações. Assim, com essas ferramentas, é possível filtrar grandes volumes de dados ou renderizar listas pesadas sem comprometer a interatividade (RAFAL, 2021).

Também foi consolidado uma nova forma para o componente *Suspense*[7] fazendo dele uma das novidades da nova arquitetura concorrente, permitindo que a interface gerencie estados de carregamento de forma mais fluida e inteligente. Diferente das abordagens antigas, em que componentes precisavam controlar manualmente *loading states*, o *Suspense* delega ao próprio *React* a responsabilidade de decidir o melhor momento para exibir, ocultar ou adiar partes da *interface*. Este comportamento é otimizado quando usado junto com a API de transição *startTransition*, se um componente suspende durante uma transição, o *React* evita substituir imediatamente o conteúdo visível por um *fallback*, e aguarda até que dados suficientes estejam prontos para evitar um estado de carregamento ruim.

Essa estratégia aprimora a experiência do usuário em cenários nos quais partes da *User Interface* (UI) dependem de dados assíncronos, evitando flashes de carregamento desnecessários e transições abruptas. Além disso, o *Suspense* tornou-se um recurso essencial em aplicações modernas, pois possibilita interfaces mais responsivas mesmo em ambientes com latência variável, alinhando-se às práticas de *User Experience* (UX) resiliente em sistemas web contemporâneos (WIERUCH, 2022).

Na área de estilização, há a possibilidade de criar estilos diretamente em código *JavaScript*, em vez de escrever arquivos CSS. Para isso, costuma-se utilizar uma biblioteca de CSS-em-*JavaScript*. Na versão 18, o *React* trouxe uma atualização importante para esse ecossistema: a criação do *hook useInsertionEffect*[8]. Esse *hook*

[6] <https://legacy.reactjs.org/blog/2022/03/29/react-v18.html>
[7] <https://react.dev/reference/react/Suspense>
[8] <https://react.dev/reference/react/useInsertionEffect>

permite que a injeção de estilos ocorra no momento exato do ciclo de renderização, antes do cálculo do *layout*, evitando, assim, *reflows* custosos em ambientes com renderização concorrente.

Esse *hook* foi projetado como uma alternativa segura ao *useLayoutEffect* em casos em que a ordem de inserção de estilos é crítica para evitar o chamado *layout thrashing*. Isso ocorre porque, se o desenvolvedor inserir estilos durante a renderização e o *React* estiver processando uma atualização não bloqueante, o navegador recalculará os estilos a cada quadro enquanto renderiza a árvore de componentes, o que pode ser extremamente lento. O *useInsertionEffect* é mais adequado do que inserir estilos dentro de *useLayoutEffect* ou *useEffect*, pois garante que, quando outros *Effects* forem executados nos componentes, as tags de estilo já tenham sido inseridas. Caso contrário, cálculos de *layout* em *Effects* comuns estariam incorretos devido a estilos desatualizados [9].

A evolução apresentada na versão 18 ganhou maturidade e novos desdobramentos com o lançamento do *React* 19, divulgado oficialmente em dezembro de 2024. Essa versão aprofunda o compromisso da biblioteca com experiências mais fluidas e com uma integração mais sólida entre cliente e servidor. Um dos avanços mais populares é a introdução das *Actions*, que representam uma nova abordagem para lidar com operações assíncronas, oferecendo controle automático de erros, gerenciamento de estado pendente e suporte nativo a atualizações (NOVOTNY, 2024).

Essa funcionalidade possibilita interfaces que reagem instantaneamente, reduzindo a latência percebida pelo usuário e minimizando retrabalhos no gerenciamento manual de estados assíncronos. Assim, aplicações que dependem de inserções, edições ou consultas dinâmicas, como o sistema que será proposto mais adiante neste trabalho, no Capítulo 3, beneficiam-se diretamente da previsibilidade e da robustez dessas novas APIs [10].

Paralelamente, o *React* 19 consolidou o modelo de *Server Components* como principal atualização. Ele já vinha sendo explorado de forma experimental na versão

[9] <https://react.dev/reference/react/useLayoutEffect>
[10] <https://react.dev/blog/2024/12/05/react-19>

18, mas apenas agora alcançou maturidade e adoção oficial. Esse avanço reduz significativamente o volume de *JavaScript* enviado ao navegador e melhora o tempo de carregamento inicial. A versão 19 também introduziu novas diretivas, como *use client* e *use server*, que delimitam de maneira explícita o contexto de execução dos módulos, simplificando a arquitetura de aplicações híbridas (GEEKSFORGEEKS, 2025).

Além disso, as novas APIs de pré-renderização presentes em *react-dom/static*, como *prerender* e *prerenderToNodeStream*, permitem a geração de HTML estático de maneira incremental via *streaming*, promovendo uma hidratação mais eficiente e uma experiência progressiva, bastante relevante para aplicações que necessitam de otimização para mecanismos de busca e carregamentos iniciais rápidos (NALAWADE, 2024).

## 2.3 Fundamentação Teórica

O *React* se consolidou como um marco no desenvolvimento web moderno por adotar uma filosofia declarativa e baseada em componentes, permitindo que os desenvolvedores descrevam o resultado final desejado da interface sem se preocupar com cada passo da atualização do DOM (BANKS; PORCELLO, 2020). No paradigma declarativo, o programador especifica o que a interface deve exibir para determinado estado da aplicação, enquanto o *React*, por meio de seu algoritmo de reconciliação e do *Virtual*DOM, calcula automaticamente como transformar o estado atual no desejado, aplicando apenas as mudanças mínimas necessárias.

Em contraste, o paradigma imperativo do *Javascript* pede que o desenvolvedor detalhe cada passo de manipulação do DOM, tornando aplicações complexas mais propensas a erros, difíceis de manter e menos escaláveis (FLANAGAN, 2020). Assim, a abordagem declarativa do *React* simplifica o gerenciamento de estado e promove previsibilidade, modularidade e manutenção facilitada, especialmente em aplicações de grande escala (MINNICK, 2022).

O *Virtual*DOM é uma das inovações centrais que permitem ao *React* funcionar

de forma eficiente. Ele atua como uma representação intermediária da interface, calculando as mudanças mínimas necessárias antes de aplicar qualquer modificação ao DOM real (BANKS; PORCELLO, 2020). Essa abstração reduz drasticamente o custo de renderizações repetidas, tornando o *React* particularmente adequado para interfaces dinâmicas e altamente interativas. Como observa ZAKAS (2016), operações diretas sobre o DOM são significativamente mais custosas, e a abordagem do *React* oferece um equilíbrio entre desempenho e simplicidade, mesmo em aplicações de grande porte.

# Capítulo 3

# SIGMEV

A necessidade de acesso tecnicamente eficiente, organizado e confiável a informações sobre medicamentos de uso veterinário, discutida no Capítulo 1, evidencia algumas lacunas significativas na prática clínica. A integração de conjuntos de dados díspares requer recursos computacionais e expertise significativos, que muitas vezes não estão disponíveis em clínicas veterinárias menores ou com recursos limitados (VLASIOU, 2024).

Nesse contexto, torna-se imprescindível adotar uma solução tecnológica capaz de centralizar informações essenciais, otimizar o fluxo de busca e reduzir a dependência de materiais dispersos ou desatualizados. Assim, o desenvolvimento de uma plataforma digital especializada surge como uma possível resposta ao problema identificado, alinhando-se às necessidades profissionais e acadêmicas da área.

Diante dessa motivação, este capítulo apresenta o Sistema de Gerenciamento de Medicamentos Veterinários (SIGMEV), proposto como uma ferramenta capaz de consolidar dados técnicos, facilitar processos de consulta e oferecer uma interface moderna, responsiva e intuitiva, utilizando a tecnologia *React*, conforme discutido no Capítulo 2. Como objetivo central, o projeto busca organizar, atualizar e disponibilizar informações detalhadas sobre medicamentos de uso veterinário, contribuindo para a precisão das decisões terapêuticas e apoiando a formação de estudantes e profissionais da área.

Para isso, o SIGMEV integra mecanismos de busca, categorização por princípios ativos, classes farmacológicas, espécies indicadas, observações importantes e outras variáveis essenciais ao uso responsável de fármacos. Ao sistematizar tais elementos em uma única plataforma, o sistema busca reduzir o tempo gasto em consultas dispersas, mitigar erros de interpretação e ajudar nas práticas veterinárias, respondendo à necessidade de integração de dados apontada por Lustgarten et al. (2020). As próximas seções detalharão o processo de concepção do SIGMEV, abordando a visão geral do sistema, trabalhos relacionados, requisitos, casos de uso e a modelagem de dados.

## 3.1 Visão geral do Sistema

O SIGMEV foi concebido para integrar o ecossistema veterinário, estruturando-se em duas vertentes operacionais: o gerenciamento da base de dados e a consulta pública. Para as rotinas de gestão e curadoria (parte privada), o sistema foi projetado para atender a três perfis de usuários: administradores (incluindo docentes e pesquisadores), profissionais da saúde e alunos (profissionais em formação).

Os administradores são os únicos responsáveis pelo cadastro de novos usuários na plataforma e, juntamente com os profissionais da saúde, possuem permissão para realizar o gerenciamento completo de medicamentos, princípios ativos, espécies e classes farmacológicas. Os alunos, por sua vez, terão permissões limitadas: poderão apenas adicionar novos medicamentos e visualizar os já cadastrados. Os medicamentos inseridos por alunos, assim como todos os recém cadastrados, permanecerão com visibilidade privada até que um administrador ou profissional da saúde revise suas informações e autorize sua publicação.

A parte pública poderá ser acessada por profissionais da saúde, incluindo alunos em formação, desde que aceitem o termo de uso que confirma sua atuação ou formação na área. Após concordar com o termo, o usuário poderá consultar medicamentos pelo nome ou princípio ativo e visualizar as informações associadas a cada item cadastrado no sistema.

Em resumo, o sistema fundamenta-se na separação funcional entre o ambiente de gestão e o ambiente de consulta. A área privada atua como um mecanismo de controle, onde a informação é inserida e validada por usuários autenticados. A área pública serve exclusivamente como interface de disseminação do conhecimento consolidado. Essa distinção assegura que apenas dados revisados e aprovados estejam disponíveis para consulta externa, preservando a integridade da base farmacológica e garantindo que o acesso a informações sensíveis ocorra mediante a devida confirmação de responsabilidade profissional.

## 3.2 Trabalhos Relacionados

A análise de soluções existentes no mercado é uma etapa fundamental para validar a relevância do SIGMEV e identificar as lacunas que o projeto se propõe a preencher. No cenário atual da informação farmacológica veterinária, duas soluções polarizam o mercado, sendo elas o *Plumb's*, que é a referência global de conteúdo, e o *Vetsmart*, que atua como plataforma de gestão predominante no Brasil. A comparação com esses agentes permite evidenciar como o SIGMEV inova ao promover a democratização do acesso e a curadoria científica colaborativa.

### 3.2.1 Plumb's

O *Plumb's* é considerado o padrão ouro em farmacologia veterinária mundial e está disponível em formatos impresso e digital (Figura 3.1). A ferramenta oferece dados exaustivos sobre farmacocinética, farmacodinâmica e dosagens. Sua confiabilidade resulta de sua rigorosa curadoria de dados, potencializada pelo meio digital. A plataforma de assinatura é constantemente revisada e atualizada por um corpo de mais de 200 especialistas globais, permitindo que as informações sobre dosagens específicas, interações medicamentosas e novas pesquisas sejam incorporadas em tempo real (PLUMB´S, 2025).

Entretanto, a aplicação dessa ferramenta no contexto acadêmico brasileiro encontra barreiras significativas que o SIGMEV visa mitigar diretamente. A primeira

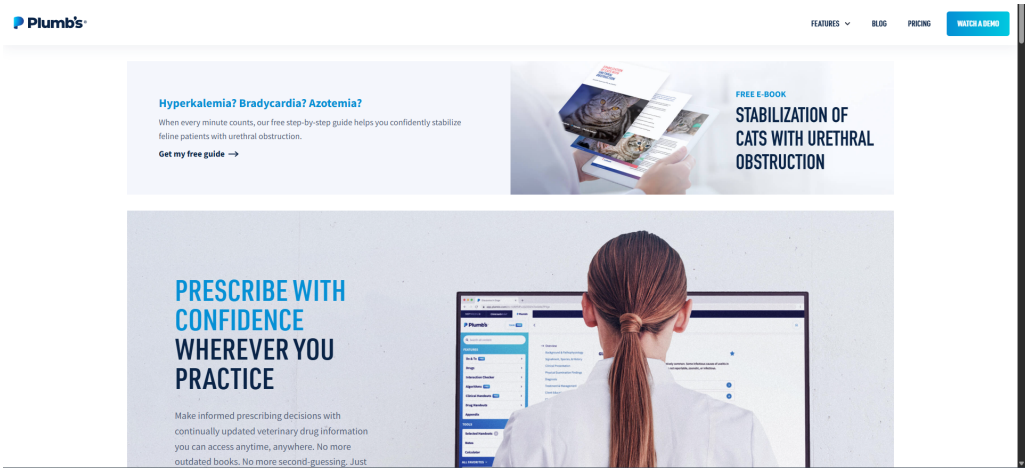


Figura 3.1: Página inicial do Plumb's.

dificuldade é o idioma, visto que a plataforma é exclusivamente em inglês e limita o acesso de estudantes e técnicos que não dominam a língua. A segunda barreira é financeira, dado o alto custo da assinatura em moeda estrangeira. O SIGMEV se diferencia ao atuar justamente na democratização do conhecimento, pois oferece informação técnica de alta qualidade em língua portuguesa e acessível a profissionais da saúde e instituições de ensino.

### 3.2.2 Vetsmart

A plataforma *Vetsmart* [1] (Figura 3.2) consolidou-se como um vasto ecossistema de gestão clínica e relacionamento com a indústria farmacêutica. Seu bulário é integrado a ferramentas de prescrição e calculadoras, o que o torna extremamente útil para a rotina comercial de consultórios.

O diferencial do SIGMEV em relação a essa solução reside na origem e no propósito da informação. Enquanto o foco do *Vetsmart* prioriza a amplitude de produtos comerciais e funcionalidades de gestão, o SIGMEV foca na profundidade científica e na independência acadêmica. Diferente de um catálogo de produtos guiado por parcerias de mercado, o sistema proposto constrói uma base de dados onde a inclusão de informações sobre fármacos é guiada estritamente por critérios farmacológicos e de necessidade clínica.

[1] <https://vetsmart.com.br/>

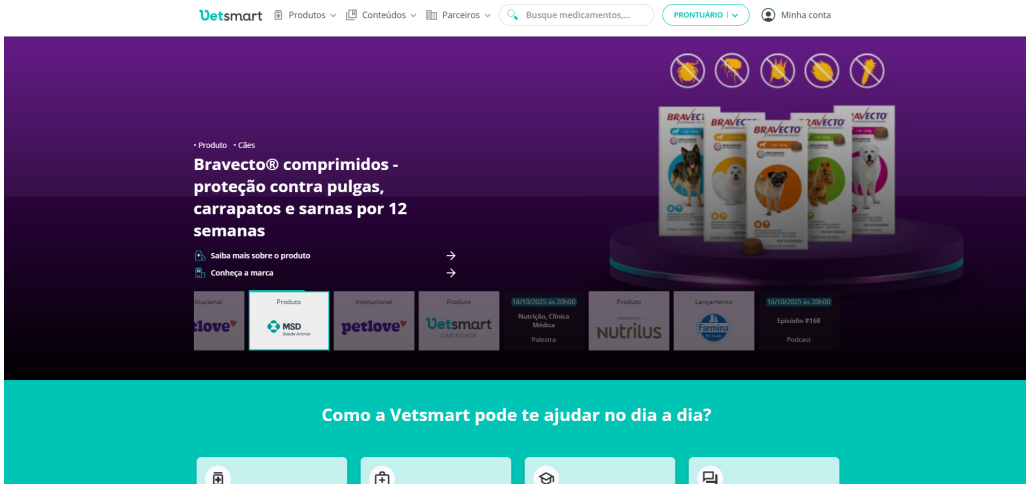


Figura 3.2: Página inicial do Vetsmart.

### 3.2.3 Posicionamento do SIGMEV

O SIGMEV se destaca pela curadoria institucional colaborativa, pois promove a validação de dados por uma rede de pesquisadores e docentes, garantindo que a informação reflita o consenso científico. Além disso, o SIGMEV preenche uma lacuna crítica ao catalogar com rigor medicamentos de uso humano frequentemente utilizados na veterinária de forma *off label*, centralizando informações que hoje se encontram dispersas. Dessa forma, o SIGMEV estabelece seu diferencial ao unir a confiabilidade da curadoria acadêmica com a soberania e a acessibilidade necessárias para a formação dos profissionais brasileiros.

## 3.3 Requisitos do Sistema

O levantamento e a especificação de requisitos constituem uma das etapas mais críticas da Engenharia de *Software*, pois definem formalmente o que o sistema deve realizar e as restrições que devem ser respeitadas (PRESSMAN, 2010). A especificação de requisitos atua como um contrato entre a equipe de desenvolvimento e os *stakeholders*, reduzindo ambiguidades e servindo de base para as atividades de verificação e validação (SOMMERVILLE, 2016). Erros cometidos nesta fase são especialmente mais difíceis de corrigir em estágios posteriores do projeto, o que reforça a necessidade de uma análise cuidadosa e detalhada (PRESSMAN, 2010).

### 3.3.1 Requisitos Funcionais

Os Requisitos Funcionais (RF) descrevem as funcionalidades e serviços que o sistema deve fornecer, incluindo o modo como ele deve reagir a determinadas entradas e comportar-se em situações específicas (SOMMERVILLE, 2016). Esses requisitos traduzem as necessidades dos usuários em comportamentos observáveis do software, definindo o que o sistema deve realizar para atingir seus objetivos.

Os seguintes RF foram definidos para o SIGMEV:

- **RF-01:** O sistema deve permitir cadastrar usuários.
- **RF-02:** O sistema deve permitir editar usuários.
- **RF-03:** O sistema deve permitir excluir usuários.
- **RF-04:** O sistema deve controlar o acesso às funcionalidades baseado no papel de cada usuário.
- **RF-05:** O sistema deve permitir o cadastro de medicamentos.
- **RF-06:** O sistema deve permitir a edição de medicamentos.
- **RF-07:** O sistema deve permitir a exclusão de medicamentos.
- **RF-08:** O sistema deve permitir a busca de medicamentos.
- **RF-09:** O sistema deve permitir a consulta dos detalhes de um medicamento.
- **RF-10:** O sistema deve permitir editar princípios ativos.
- **RF-11:** O sistema deve permitir excluir princípios ativos.
- **RF-12:** O sistema deve permitir cadastrar classes farmacológicas.
- **RF-13:** O sistema deve permitir editar classes farmacológicas.
- **RF-14:** O sistema deve permitir excluir classes farmacológicas.
- **RF-15:** O sistema deve permitir cadastrar espécies animais.

- **RF-16:** O sistema deve permitir editar espécies animais.
- **RF-17:** O sistema deve permitir excluir espécies animais.

### 3.3.2 Regras de Negócio

As Regras de Negócio (RN) são diretrizes que governam o funcionamento do sistema, refletindo políticas, procedimentos e restrições do domínio veterinário. Diferentemente dos RF e Requisitos Não Funcionais (RNF), elas não descrevem o que o sistema faz ou como se comporta tecnicamente, mas determinam as condições e restrições operacionais que garantem a consistência, integridade e validade dos dados e processos.

As seguintes RN foram definidos para o SIGMEV:

- **RN-01 - Aplica-se ao RF-01 e RF-02:** O cadastro de novos usuários e edição só podem ser realizados por um administrador. É necessário o preenchimento dos seguintes campos obrigatórios: “nome completo”, “e-mail”, “papel” e “senha”. Um usuário deve possuir um dos seguintes papéis: Administrador, Profissional da Saúde ou Aluno.
- **RN-02 - Aplica-se ao RF-03:** A exclusão de um usuário só pode ser feita por um administrador e deve exigir confirmação explícita do mesmo, escrevendo o e-mail do usuário a ser excluído.
- **RN-03 - Aplica-se ao RF-04:** O acesso às funcionalidades é restrito por papel. Apenas Administradores podem gerenciar usuários (RF-01, RF-02, RF-03). Apenas Administradores e Profissionais da Saúde podem gerenciar medicamentos (RF-05, RF-06, RF-07), princípios ativos (RF-10, RF-11), classes farmacológicas (RF-12, RF-13, RF-14) e espécies (RF-15, RF-16, RF-17). Os Alunos podem apenas cadastrar medicamentos (RF-05) e visualizar a lista de medicamentos cadastrados.
- **RN-04 - Aplica-se ao RF-05:** O cadastro e edição de medicamentos exige os seguintes campos obrigatórios: “Nome comercial”, “Princípio ativo”, “Classe

farmacológica", "Espécie", "Descrição do medicamento", "Posologia", "Indicação de uso", "Restrição de uso" e "Observações importantes".

- **RN-05 - Aplica-se ao RF-05 e RF-06:** Todos os medicamentos recém cadastrados devem ter visibilidade privada, podendo ser alterada para público por um Profissional da Saúde ou Administrador.

- **RN-06 - Aplica-se ao RF-07:** A exclusão de um medicamento deve exigir confirmação explícita do usuário.

- **RN-07 - Aplica-se ao RF-08:** Para buscar um medicamento (RF-08), o usuário deve primeiro aceitar um termo de condição confirmando que é um profissional da saúde.

- **RN-08 - Aplica-se ao RF-08:** A busca de medicamentos (RF-08) é pública por nome comercial ou princípio ativo.

- **RN-09 - Aplica-se ao RF-10:** A edição de um princípio ativo (RF-10) limita-se à alteração do seu nome. O cadastro de novos princípios ativos ocorre exclusivamente através do cadastro de medicamentos (RF-05), não sendo permitido um cadastro avulso.

- **RN-10 - Aplica-se ao RF-11:** A exclusão de um princípio ativo (RF-11) só é permitida se o princípio ativo não estiver associado a nenhum medicamento; caso contrário, a operação deve ser negada.

- **RN-11 - Aplica-se ao RF-12 e RF-13:** O cadastro e a edição de classes farmacológicas exigem apenas a informação do nome da classe.

- **RN-12 - Aplica-se ao RF-14:** A exclusão de uma classe farmacológica só é permitida se esta não estiver associada a nenhum medicamento; caso contrário, a operação deve ser negada.

- **RN-13 - Aplica-se ao RF-15 e RF-16:** O cadastro e a edição de espécies exigem apenas a informação do nome da espécie.

- **RN-14 - Aplica-se ao RF-17:** A exclusão de uma espécie só é permitida se esta não estiver associada a nenhum medicamento; caso contrário, a operação deve ser negada.

### 3.3.3 Casos de Uso

Os Casos de Uso (CU) descrevem como os diferentes atores interagem com o sistema para alcançar objetivos específicos, representando o comportamento funcional esperado do *software* sob a perspectiva do usuário final (SOMMERVILLE, 2016). Cada caso de uso detalha as ações principais, as condições de entrada e saída, e as possíveis exceções que podem ocorrer durante a execução.

O Diagrama de Casos de Uso do sistema (Figura 3.3) ilustra as interações funcionais e a hierarquia de permissões entre os atores envolvidos. O Administrador possui acesso total ao sistema: ele herda todas as competências de gestão de informações farmacológicas (e.g., medicamentos, princípios ativos, espécies, classes farmacológicas) e, adicionalmente, detém a exclusividade sobre a manutenção administrativa, como o cadastro, edição e exclusão de usuários.

Por sua vez, o ator Profissional da Saúde atua no núcleo operacional do sistema, concentrando-se estritamente no gerenciamento dos dados técnicos veterinários, sem permissões para alterar credenciais de acesso de outros usuários. Os atores Aluno e Usuário Público possuem escopos de interação mais restritos, limitando-se, respectivamente, à colaboração no cadastro de medicamentos e à consulta de informações. O detalhamento completo de todos os fluxos, encontra-se na Especificação de Casos de Uso, disponível no Apêndice deste trabalho.

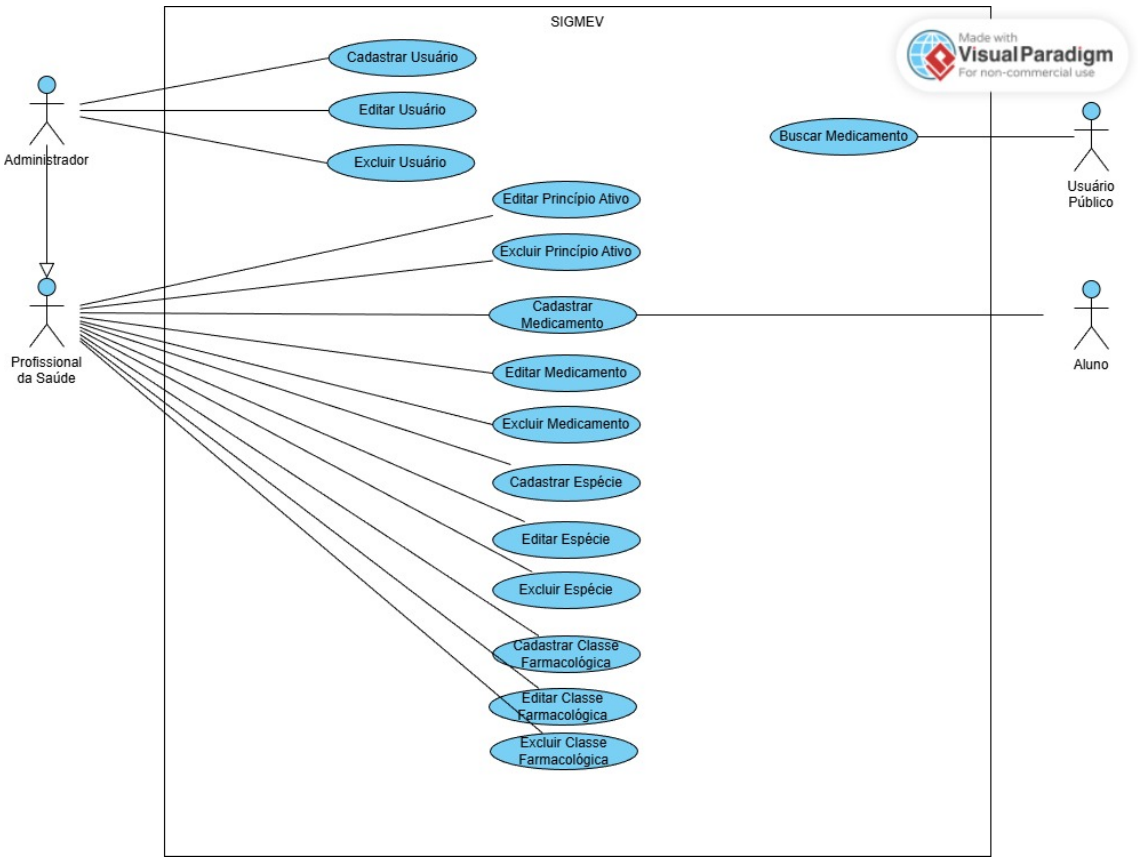


Figura 3.3: Diagrama de casos de uso.

## 3.4 Modelagem de Dados

Para representar a estrutura lógica dos dados do sistema e as relações entre as principais entidades, foi elaborado um Diagrama Entidade Relacionamento (DER). Esse diagrama tem como objetivo demonstrar como as informações são organizadas, armazenadas e interligadas no banco de dados, garantindo consistência, integridade e facilidade de manutenção das informações referentes aos medicamentos de uso veterinário e seus respectivos atributos.

A modelagem do banco de dados é uma etapa essencial, pois define a estrutura que sustentará todas as operações do sistema, como o cadastro, edição e consulta de medicamentos. O projeto lógico de um banco de dados relacional tem como objetivo

eliminar redundâncias, assegurar integridade e fornecer uma estrutura eficiente para o armazenamento e a recuperação dos dados (DATE, 2004).

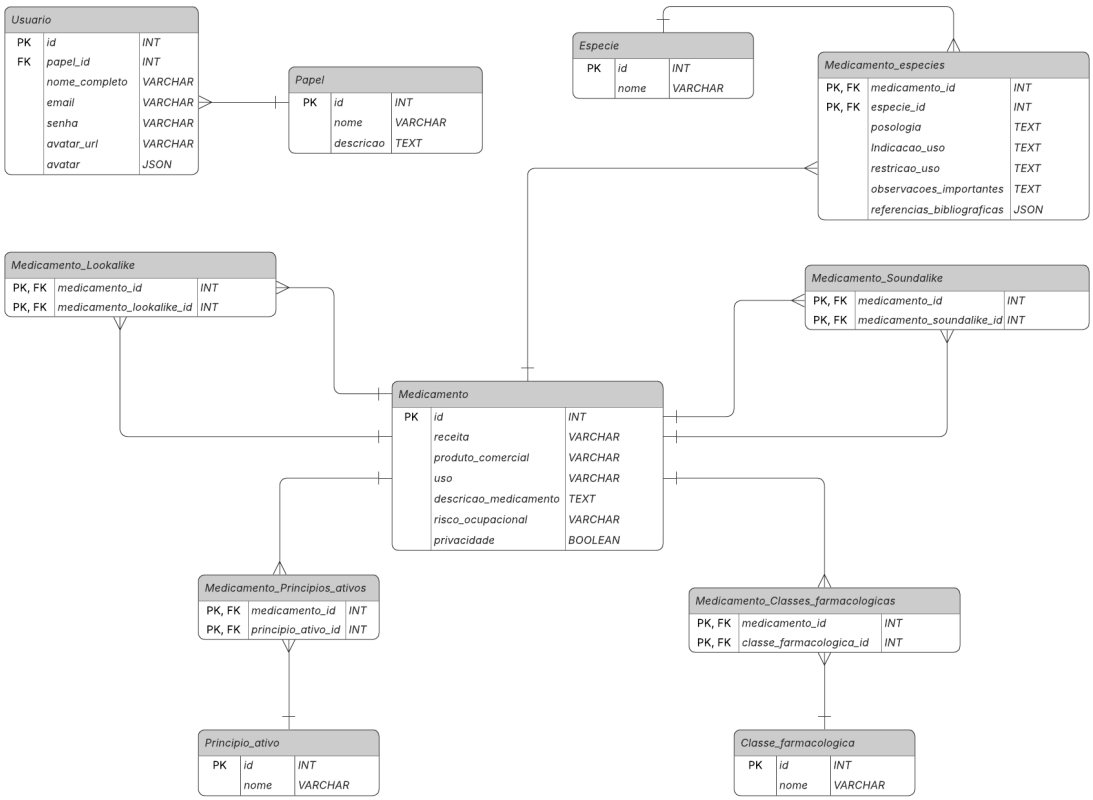


Figura 3.4: Diagrama Entidade-Relacionamento do SIGMEV.

O modelo apresentado na Figura 3.4 foi concebido para atender a todos os requisitos funcionais e regras de negócio levantados, garantindo a integridade referencial dos dados. Sua estrutura foi pensada para oferecer uma base consistente e escalável, capaz de sustentar futuras expansões do sistema.

### 3.4.1 Entidades

A entidade *Usuário* (Tabela 3.1) é responsável por armazenar as informações de identificação e acesso dos usuários do sistema, como nome, e-mail, senha criptografada e status de atividade. Essa entidade é fundamental para o controle de autenticação e autorização no ambiente privado do sistema, atendendo diretamente aos RF-01,

RF-02 e RF-03. Possui Chave Primária (PK) para identificação única na tabela, podendo ser referenciada como Chave Estrangeira (FK) para outras entidades.

Já a entidade *Papel* (Tabela 3.2) define os diferentes perfis de acesso disponíveis na plataforma, conforme especificado na RN-01. Cada função possui permissões distintas, garantindo a segurança e a integridade dos dados manipulados e viabilizando o cumprimento do RF-04.

Tabela 3.1: Estrutura da Tabela `Usuario`.

| Coluna | Descrição |
|---|---|
| `id` PK | Identificador único do usuário. |
| `role_id` FK | Chave estrangeira que referencia a tabela `Papel`. |
| `nome_completo` | Nome completo do usuário. |
| `e-mail` | Endereço de e-mail para login (único). |
| `senha` | Senha criptografada do usuário. |
| `avatar_url` | URL para a imagem de avatar do usuário. |
| `avatar` | JSON para armazenar metadados do avatar. |

Tabela 3.2: Estrutura da Tabela `Papel`.

| Coluna | Descrição |
|---|---|
| `id` PK | Identificador único do papel (perfil). |
| `nome` | Nome do papel (e.g., Administrador, Aluno). |
| `descricao` | Descrição das permissões do papel. |

A entidade *Medicamento* (Tabela 3.3) é a principal do modelo proposto. Ela é responsável por armazenar informações gerais sobre os medicamentos de uso veterinário, como nome comercial, tipo de receita, modo de uso, descrição, risco ocupacional e visibilidade (pública ou privada). Esta tabela é a base para a execução dos requisitos RF-05 (Cadastrar Medicamento), RF-06 (Editar Medicamento) e RF-07 (Excluir Medicamento). Os campos obrigatórios para o cadastro e edição estão definidos na RN-04.

A entidade *Principio_ativo* (Tabela 3.4) armazena os dados referentes aos componentes químicos responsáveis pelos efeitos farmacológicos dos medicamentos. Essa entidade foi modelada de forma independente para atender aos requisitos RF-10 e RF-11 (Editar e Excluir princípios ativos) e às regras de negócio RN-09 e RN-10, que define suas regras de manipulação. Da mesma forma, a entidade *Classe_farmacologica*

Tabela 3.3: Estrutura da Tabela `Medicamento`.

| Coluna | Descrição |
|---|---|
| `id` PK | Identificador único do medicamento. |
| `receita` | Tipo de receita exigida para o medicamento. |
| `produto_comercial` | Nome comercial do medicamento. |
| `uso` | Modo de uso do medicamento. |
| `descricao_medicamento` | Descrição detalhada do medicamento. |
| `risco_ocupacional` | Informações sobre risco ocupacional. |
| `privacidade` | Controle de visibilidade (Público/Privado). |

(Tabela 3.5) tem como objetivo classificar os medicamentos de acordo com sua finalidade terapêutica. Essa estrutura implementa os requisitos RF-12, RF-13 e RF-14, e segue as regras RN-11 e RN-12.

Tabela 3.4: Estrutura da Tabela `Principio_ativo`.

| Coluna | Descrição |
|---|---|
| `id` PK | Identificador único do princípio ativo. |
| `nome` | Nome do princípio ativo. |

Tabela 3.5: Estrutura da Tabela `Classe_farmacologica`.

| Coluna | Descrição |
|---|---|
| `id` PK | Identificador único da classe farmacológica. |
| `nome` | Nome da classe farmacológica. |

Adicionalmente, a entidade *Especie* (Tabela 3.6) foi modelada para representar as diferentes espécies animais às quais um medicamento pode ser indicado, garantindo que as informações sobre o uso sejam precisas e contextualizadas. Essa modelagem atende aos requisitos RF-14, RF-15 e RF-16, também às regras RN-13 e RN-14, permitindo especificar, de forma estruturada, quais medicamentos são adequados para cada espécie, evitando prescrições incorretas e contribuindo para a segurança e a eficácia dos tratamentos veterinários.

Tabela 3.6: Estrutura da Tabela `Especie`.

| Coluna | Descrição |
|---|---|
| `id` PK | Identificador único da espécie animal. |
| `nome` | Nome da espécie (ex: Canina, Felina). |

### 3.4.2 Relacionamentos

O relacionamento entre as entidades é fundamental para a correta organização e o manejo das informações no sistema. No modelo proposto, a entidade *Medicamento* possui um relacionamento muitos-para-muitos com *Principio_ ativo*, uma vez que um medicamento pode conter diversos princípios ativos, e um princípio ativo pode estar presente em vários medicamentos. Para representar esse relacionamento, foi criada a tabela associativa *Medicamento_ Principios_ ativos* (Tabela 3.7), contendo as chaves estrangeiras de ambas as entidades.

Tabela 3.7: Estrutura da Tabela Associativa `Medicamento_Principios_ativos`.

| Coluna | Descrição |
|---|---|
| `medicamento_id` PK, FK | Chave estrangeira que referencia a tabela `Medicamento`. |
| `principio_ativo_id` PK, FK | Chave estrangeira que referencia a tabela `Principio_ativo`. |

De maneira semelhante, a entidade *Medicamento* se relaciona com *Especie*, também por meio de uma relação muitos-para-muitos, pois um mesmo medicamento pode ser indicado para diferentes espécies animais. Esse relacionamento possui atributos específicos, como posologia e restrição de uso, já que essas informações variam conforme o tamanho, metabolismo ou tipo da espécie. Essa relação e seus atributos são representados pela tabela intermediária *Medicamento_ especies* (Tabela 3.8).

Tabela 3.8: Estrutura da Tabela Associativa `Medicamento_especies`.

| Coluna | Descrição |
|---|---|
| `medicamento_id` PK, FK | Chave estrangeira que referencia a tabela `Medicamento`. |
| `especie_id` PK, FK | Chave estrangeira que referencia a tabela `Especie`. |
| `posologia` | Instruções de dosagem específicas para esta espécie. |
| `indicacao_uso` | Indicação de uso específica para esta espécie. |
| `restricao_uso` | Restrições de uso específicas para esta espécie. |
| `observacoes_importantes` | Observações relevantes para a espécie. |
| `referencias_bibliograficas` | JSON para armazenar referências da posologia. |

Da mesma forma, *Medicamento* e *Classe_ farmacologica* possuem uma relação

muitos-para-muitos, visto que um medicamento pode pertencer a mais de uma classe terapêutica e cada classe pode englobar diversos medicamentos. Para representar essa associação, foi criada a tabela intermediária *Medicamento_Classes_farmacológicas* (Tabela 3.9), responsável por vincular as chaves primárias das duas entidades.

Tabela 3.9: Estrutura da Tabela Associativa *Medicamento_Classes_farmacologicas*.

| Coluna | Descrição |
|---|---|
| `medicamento_id` PK, FK | Chave estrangeira que referencia a tabela `Medicamento`. |
| `classe_farmacologica_id` PK, FK | Chave estrangeira que referencia a tabela `Classe_farmacologica`. |

Por fim, os relacionamentos que envolvem as entidades *Lookalike* e *Soundalike* são do tipo muitos-para-muitos autorreferenciais, pois registram casos em que dois ou mais medicamentos apresentam semelhanças visuais ou fonéticas entre si. Dessa forma, um medicamento pode ser semelhante a vários outros e, simultaneamente, ser referenciado por eles.

Na farmacologia veterinária, a precisão na identificação do princípio ativo é crítica, pois a fisiologia comparada dita que um medicamento seguro para uma espécie pode ser ineficaz ou letal para outra devido a diferenças metabólicas e idiossincrasias. Um exemplo envolve os corticosteroides Prednisona e Prednisolona. A confusão ortográfica entre esses compostos pode levar à administração de Prednisona em felinos ou equinos, espécies que possuem baixa eficiência hepática para converter este profármaco em sua forma ativa (Prednisolona), resultando em falha terapêutica. Analogamente, erros fonéticos entre antibióticos e imunossupressores com nomes semelhantes podem acarretar intoxicações severas.

Para representar essas relações, foram criadas as tabelas associativas *Medicamento_Lookalike* (Tabela 3.10) e *Medicamento_Soundalike* (Tabela 3.11), que contêm as chaves estrangeiras da própria entidade *Medicamento*. Uma dessas chaves aponta para o medicamento principal e outra para o medicamento semelhante.

Tabela 3.10: Estrutura da Tabela Associativa `Medicamento_Lookalike`.

| Coluna | Descrição |
|---|---|
| `medicamento_id` PK, FK | Chave do medicamento principal. |
| `medicamento_lookalike_id` PK, FK | Chave do medicamento com aparência similar. |

Tabela 3.11: Estrutura da Tabela Associativa `Medicamento_Soundalike`.

| Coluna | Descrição |
|---|---|
| `medicamento_id` PK, FK | Chave do medicamento principal. |
| `medicamento_soundalike_id` PK, FK | Chave do medicamento com fonética similar. |

# Capítulo 4

# Implementação

Este capítulo apresenta de forma detalhada o processo de construção do SIGMEV, enfatizando as etapas de desenvolvimento e as escolhas realizadas ao longo do projeto. Serão destacadas as principais tecnologias adotadas, justificando sua utilização frente às necessidades identificadas, além do registro das decisões técnicas tomadas em cada fase para alcançar os objetivos propostos.

## 4.1 Tecnologias Utilizadas

A escolha das tecnologias utilizadas neste trabalho considerou diversos fatores relevantes para o desenvolvimento de um sistema moderno e eficiente. Foram analisadas características relacionadas à eficiência, confiabilidade, facilidade de manutenção e adaptabilidade do sistema frente a novas demandas. Também se levou em conta a capacidade de integração entre diferentes módulos, a consistência no desempenho e a possibilidade de evolução futura da aplicação. Dessa forma, as tecnologias selecionadas oferecem uma base sólida, permitindo que o sistema atenda aos objetivos propostos de maneira consistente ao longo do tempo.

### 4.1.1 React

O *React*[1], como citado anteriormente no capítulo 2, é uma biblioteca *JavaScript* para a construção de interfaces de usuário web e nativas. Ele permite compor interfaces complexas a partir de pequenas e isoladas peças de código chamadas componentes. Além disso, o *React* adota uma abordagem declarativa, o que facilita o raciocínio sobre o estado da aplicação e melhora a previsibilidade do comportamento da interface.

Um dos principais fatores que motivaram a escolha do *React* para o desenvolvimento do projeto foi a sua natureza *Single Page Application* (SPA). Esse paradigma possibilita a criação de uma experiência de usuário mais fluida e contínua, eliminando a necessidade de recarregamentos completos a cada interação ou navegação entre seções.

No contexto do sistema proposto, essa característica foi especialmente relevante para a consulta de medicamentos, o cadastro e gerenciamento de usuários, bem como para a interface do painel administrativo, que exigem eficiência para garantir usabilidade e confiabilidade. Dessa forma, a utilização do *React* contribuiu não apenas para a performance técnica da aplicação, mas também para a melhoria da experiência de interação dos usuários finais.

### 4.1.2 AdonisJS

O *AdonisJS*[2] é um *framework web* para *Node.js* que prioriza o uso de *TypeScript* (*TypeScript-first*). Além disso, o *framework* pode ser utilizado tanto para a criação de aplicações *full-stack*, abrangendo *front-end* e *back-end*, quanto para o desenvolvimento de servidores de API em *JSON*. Essa versatilidade o consolida como uma solução robusta e moderna para sistemas escaláveis.

Outro ponto relevante do *AdonisJS* é a adoção do padrão de *design Model-View-Controller* (MVC). Esse padrão organiza a aplicação em camadas com responsabili-

[1] <https://react.dev/>
[2] <https://docs.adonisjs.com/guides/preface/introduction>

dades distintas, favorecendo a clareza estrutural e a manutenibilidade do sistema. No contexto do presente projeto, a utilização do MVC foi fundamental para garantir uma divisão lógica entre dados, regras de negócio e interface, o que contribuiu para o desenvolvimento de uma aplicação mais organizada e de fácil evolução.

A escolha do *AdonisJS* foi determinante para a eficiência do projeto. Sua estrutura centralizou a lógica de negócios e o processamento de dados do sistema de gerenciamento, assegurando modularidade e separação de responsabilidades. Além disso, ferramentas nativas como a *Object-Relational Mapping* (ORM) simplificaram a interação com o banco de dados, reduzindo a complexidade da manipulação de informações médicas. O uso do sistema de *migrations* também se mostrou essencial, permitindo o controle evolutivo do esquema de dados e garantindo maior flexibilidade para adaptações futuras, aspecto crucial para a manutenção e expansão de um sistema de gerenciamento de medicamentos.

### 4.1.3 InertiaJS

O *Inertia.js*[3] é uma nova abordagem para criar aplicações *web* voltadas ao servidor. Com o *Inertia*, é possível criar SPA sem a complexidade que SPAs modernas exigem. O *Inertia* não possui roteamento do lado do cliente, nem requer uma API. A abordagem funciona muito bem com *frameworks* já estabelecidos, como *Laravel*[4] e *Ruby on Rails*[5], permitindo aos desenvolvedores manter a familiaridade do desenvolvimento tradicional de aplicações monolíticas, ao mesmo tempo que oferece a experiência de usuário de uma SPA moderna.

No desenvolvimento do projeto, o uso do *Inertia.js* foi fundamental para integrar o *back-end* em *AdonisJS* com o *front-end* em *React*, criando uma ponte eficiente sem a necessidade de construir uma API separada. Essa abordagem simplificou a comunicação entre as camadas, eliminando complexidades comuns no desenvolvimento de SPAs modernas e permitindo manter uma arquitetura organizada dentro de um *monorepo*. Com isso, o fluxo de trabalho tornou-se mais ágil e produtivo, já que foi

[3] <https://inertiajs.com/>
[4] <https://laravel.com/>
[5] <https://rubyonrails.org/>

possível aproveitar o poder do *React* para a interface sem abrir mão da estrutura robusta do *AdonisJS* no *back-end*, resultando em uma aplicação com experiência de usuário fluida e consistente.

### 4.1.4 PostgreSQL

O *PostgreSQL*[6] é reconhecido como um robusto sistema de gerenciamento de banco de dados objeto-relacional de código aberto, que se destaca por utilizar e estender a linguagem *Structured Query Language* (SQL), incorporando diversas funcionalidades para armazenar e escalar com segurança as cargas de trabalho de dados mais complexas. Sua reputação é solidificada pela arquitetura comprovada, pela confiabilidade, pela integridade dos dados e pelo conjunto robusto de recursos que oferece. Adicionalmente, sua extensibilidade e a dedicação contínua da comunidade de código aberto garantem a entrega consistente de soluções inovadoras e de alto desempenho no mercado.

A escolha do *PostgreSQL* como banco de dados do projeto foi motivada por fatores como escalabilidade, extensibilidade e segurança. Além disso, o *PostgreSQL* oferece suporte a consultas complexas, indexação avançada, tipos de dados personalizados e extensões que aumentam significativamente sua flexibilidade. Outro aspecto relevante é a confiabilidade e consistência dos dados garantidas por seus mecanismos de transação (ACID), fundamentais para aplicações que exigem integridade, como o gerenciamento de medicamentos. Com essas características, o *PostgreSQL* se apresentou como a solução ideal para sustentar o crescimento do projeto, permitindo lidar com grandes volumes de informação sem comprometer o desempenho ou a segurança.

### 4.1.5 shadcn/ui

O *shadcn/ui*[7] se apresenta como um conjunto de componentes acessíveis e com um *design* elegante, atuando também como uma plataforma de distribuição de

[6] <https://www.postgresql.org/>
[7] <https://ui.shadcn.com/>

código. Sua criação visa solucionar desafios no desenvolvimento *frontend* e está fundamentada nos seguintes princípios essenciais: o código aberto, que permite a modificação da camada superior do código do componente; a composição, onde cada componente utiliza uma interface comum e combinável, assegurando previsibilidade; a distribuição, facilitada por um esquema de arquivo plano e uma ferramenta de linha de comando *Command-Line Interface* (CLI), que simplificam a distribuição dos componentes.

A utilização do *shadcn/ui* no *frontend* do SIGMEV foi deliberada e trouxe benefícios-chave para o desenvolvimento. Os componentes disponibilizados pela biblioteca foram amplamente empregados na estilização da aplicação, o que possibilitou manter uma identidade visual consistente e alinhada às boas práticas de *design* moderno. Além disso, a característica de fácil modificação do código mostrou-se essencial para adaptar os componentes às necessidades específicas do projeto, sem que fosse necessário desenvolver cada elemento do zero. Portanto, essa flexibilidade reduziu o tempo de implementação e aumentou a produtividade, permitindo que o foco permanecesse nas funcionalidades principais do sistema, ao mesmo tempo em que se assegurava uma interface agradável e acessível ao usuário final.

## 4.2 Estrutura e Arquitetura do Projeto Proposto

A definição da estrutura do projeto foi guiada pela necessidade de estabelecer uma base sólida e organizada para o desenvolvimento da aplicação. Foram considerados aspectos como a padronização do código, a facilidade de manutenção, a escalabilidade e a integração entre os módulos do sistema. Nesse contexto, optou-se pela utilização de um *starter kit*, que fornece uma configuração inicial consistente e reutilizável, facilitando o início do desenvolvimento e promovendo boas práticas desde as primeiras etapas.

Além disso, adotou-se o modelo de *monorepo*, que significa um monorepositório centralizando todos os módulos em um único repositório, permitindo melhor controle de versão, integração contínua e maior coerência entre as partes do sistema. Essa

estrutura visa garantir eficiência, coesão e evolutividade, assegurando que o projeto se mantenha sustentável e de fácil expansão ao longo do tempo.

A arquitetura do projeto proposto foi concebida com o objetivo de garantir organização e clareza na manutenção do código. A aplicação adota o padrão de *design* MVC, amplamente utilizado no desenvolvimento de sistemas web por promover a separação de responsabilidades entre as camadas de dados, interface e lógica de controle.

O projeto foi estruturado de forma modular por funcionalidade, permitindo que cada módulo represente um domínio específico do sistema, com seus próprios componentes e regras de negócio. Essa organização modular torna o sistema reutilizável e de fácil evolução, ao mesmo tempo em que possibilita uma integração harmoniosa entre as diferentes partes da aplicação.

### 4.2.1 *Starter kit*

Em muitos projetos, cada desenvolvedor depende de um conjunto comum de atividades como *design*, construção, codificação, implantação e teste para o desenvolvimento de software (SRINIVASAN et al., 2025). Percorrer essas etapas de forma repetitiva em cada novo projeto acaba por reduzir a produtividade do desenvolvedor (SRINIVASAN et al., 2025). O *starter kit* surge como uma solução para mitigar essa problemática, oferecendo vantagens relacionadas aos custos iniciais de configuração, integração e aprendizado.

Um *starter kit* pode ser entendido como um conjunto pré-configurado de ferramentas, bibliotecas e estruturas de pastas, criado com o objetivo de agilizar o início de um projeto. Dessa forma, reduz-se o tempo de preparação do ambiente e garante-se maior padronização entre os módulos do sistema. O *starter kit* pode fornecer uma série de recursos, como *templates*, *scripts* e configurações padrão. Entre as diversas vantagens da utilização do *starter kit* neste projeto, destacam-se o ganho de produtividade, a redução de erros, o aumento do padrão de qualidade do *software* e a padronização do código.

No presente trabalho, a utilização de um *starter kit* mostrou-se fundamental. Como base estrutural, foi adotado o *AdonisJS Starter Kit* [8]. Esse *starter kit* utilizou *AdonisJS* e *React* como tecnologias principais de *back-end* e *front-end*, respectivamente, além de um modelo de monorepositório (*monorepo*), já configurado para utilização prática. Também estavam inclusos mecanismos de autenticação, como *login*, cadastro e recuperação de senha. Com essa estrutura previamente estabelecida, a produtividade no desenvolvimento foi significativamente ampliada, uma vez que componentes complexos já estavam prontos para uso e adaptação ao contexto do projeto.

### 4.2.2 Monorepo

A adoção do modelo de *monorepo* tem se mostrado uma prática vantajosa em projetos de software de médio e grande porte. Essa abordagem permite maior interação e sinergia entre diferentes projetos, reduzindo a duplicação de esforços e aumentando a eficiência no desenvolvimento (SHAKIKHANLI; BILICKI, 2024). Diferentemente do modelo de múltiplos repositórios (*multirepo*), no qual cada serviço ou aplicação é mantido isoladamente, o *monorepo* centraliza todo o código-fonte em um único repositório, promovendo padronização, melhor gestão de dependências, maior facilidade de colaboração entre equipes e possibilita um melhor versionamento de código.

No presente trabalho, o *monorepo* abriga tanto a camada de *back-end*, quanto a camada de *front-end*. Ambas são interligadas pela biblioteca *InertiaJS*, que permite a comunicação entre as duas camadas sem a necessidade de uma *Representational State Transfer Application Programming Interface* (API REST), simplificando a arquitetura e otimizando a troca de dados. Além disso, o repositório contempla pacotes auxiliares, como bibliotecas de componentes reutilizáveis e definições de tipos compartilhados, o que garante maior consistência e reduz a redundância de código.

A organização e a gestão das dependências foram realizadas por meio do *pnpm*, enquanto o *Turborepo* foi empregado para a orquestração de diferentes processos,

[8] <https://github.com/filipebraida/adonisjs-starter-kit>

como a compilação do código *build*, oferecendo *cache* inteligente e a capacidade de rodar tarefas de forma eficaz. O gerenciamento eficiente das dependências e a execução paralela de tarefas reduziram o tempo necessário para etapas críticas do ciclo de desenvolvimento. Além de acelerar o processo, essa abordagem permitiu que somente os pacotes efetivamente modificados fossem reprocessados, economizando recursos computacionais.

A utilização do *monorepo* possibilitou uma maior consistência na evolução do projeto, já que tanto o *back-end* quanto o *front-end*, além das bibliotecas auxiliares, compartilham a mesma base de versionamento e processos de manutenção. Essa integração reduziu significativamente a duplicação de código e o retrabalho, uma vez que ajustes em pacotes comuns passaram a ser propagados automaticamente para todos os módulos dependentes. Dessa forma, tornou-se possível manter uma arquitetura mais coesa, aumentando a integridade do sistema.

### 4.2.3 Padrão de *design* MVC

Um dos pontos centrais do *framework AdonisJS* é a adoção do padrão MVC. Segundo Pop e Altar (2013), o MVC é amplamente utilizado no desenvolvimento de aplicações web porque combina diferentes tecnologias em um conjunto de camadas interdependentes, porém bem definidas. Esse padrão favorece a separação de responsabilidades, permitindo que cada camada seja desenvolvida, testada e mantida de forma independente. Além disso, isolar as unidades funcionais umas das outras o máximo possível torna mais fácil para o projetista da aplicação entender e modificar cada unidade particular sem precisar conhecer profundamente as demais (KRASNER; POPE, 1988).

O MVC organiza a aplicação em três camadas distintas, com papéis claramente delimitados. A camada *Model* é responsável pelo gerenciamento dos dados e pela implementação da lógica de negócio, garantindo a consistência e integridade das informações. A camada *View* concentra-se na exibição das informações, fornecendo ao usuário uma interface intuitiva e agradável. Por sua vez, a camada *Controller* atua como intermediária, processando as requisições do usuário, manipulando os

dados conforme necessário e retornando as respostas adequadas.

No contexto deste trabalho, o MVC foi aplicado de forma integral em cada módulo do sistema desenvolvido com o *AdonisJS*. A camada *Model* foi utilizada para representar as entidades do sistema e gerenciar as interações com o banco de dados, utilizando o ORM do próprio *framework*. A camada *Controller* concentrou as regras de negócio e o processamento das requisições *Hypertext Transfer Protocol* (HTTP), sendo responsável por intermediar a comunicação entre o usuário e os dados da aplicação. Já a camada *View* foi estruturada para trabalhar de forma integrada com o *React* por meio do *Inertia.js*, garantindo uma experiência fluida e interativa. Essa aplicação prática do MVC contribuiu diretamente para a organização do código e para a modularidade geral do projeto.

### 4.2.4 Arquitetura Modular por Funcionalidade

A arquitetura adotada neste projeto segue o princípio da modularização por funcionalidade, uma abordagem que organiza o sistema em unidades autônomas, cada uma responsável por um domínio específico da aplicação. Essa estrutura permite que cada módulo concentre suas próprias rotas, controladores, modelos e serviços, reduzindo o acoplamento entre componentes e favorecendo a coesão interna. No trabalho, os principais módulos criados foram *vet, users, auth e core*, cada um com funções bem definidas e independentes.

O módulo *vet* concentra todas as funcionalidades relacionadas à área veterinária, como o cadastro, listagem, pesquisa e gerenciamento de dados específicos do domínio. O módulo *users* é responsável pelas operações de registro, atualização e controle de usuários do sistema. Já o módulo *auth* gerencia os processos de autenticação e autorização, garantindo a segurança e o controle de acesso às rotas e recursos. O módulo *core* atua como a base estrutural do sistema, fornecendo recursos e configurações que são compartilhados entre os demais módulos.

Essa abordagem modular traz uma série de benefícios para o desenvolvimento e manutenção do sistema. Primeiramente, facilita a escalabilidade, uma vez que novos módulos podem ser adicionados sem afetar a estrutura dos existentes. Além disso,

promove uma maior organização do código, tornando mais simples a compreensão e o rastreamento de funcionalidades específicas. Cada módulo pode ser mantido e atualizado de forma isolada, o que reduz o risco de introdução de erros em outras partes da aplicação.

Outro ponto relevante dessa abordagem é a facilidade na realização de testes. A divisão do sistema em módulos independentes permite que cada parte seja testada de forma isolada, o que aumenta a precisão e a confiabilidade dos resultados. Com essa estrutura, é possível aplicar testes unitários e de integração de maneira mais eficiente, validando as funcionalidades de cada módulo separadamente antes de integrá-los ao sistema principal. Isso não apenas reduz o tempo necessário para identificar falhas, como também simplifica a manutenção e a evolução do código, tornando o processo de desenvolvimento mais seguro e controlado.

Portanto, a adoção da arquitetura modular por funcionalidade neste trabalho justifica-se por sua capacidade de oferecer uma base sólida e escalável para o sistema desenvolvido. Essa estrutura favorece a clareza na organização do código, melhora a legibilidade, facilita a testagem e contribui diretamente para a qualidade do produto final. Além disso, ao promover a separação de responsabilidades e o isolamento entre componentes, essa abordagem assegura maior flexibilidade para futuras expansões e atualizações, consolidando-se como uma solução eficiente e sustentável para o desenvolvimento da aplicação *web*.

## 4.3 Interfaces do Sistema

O *design* da interface do sistema foi fundamentado em princípios de UX, com o objetivo de criar uma ferramenta que fosse não apenas funcional, mas também intuitiva e eficiente ao objetivo. Para isso, foram desenvolvidas telas distintas para os dois principais casos de uso: a consulta pública e a gestão administrativa.

### 4.3.1 Fluxo de Consulta Pública

O fluxo de consulta se inicia na tela principal do SIGMEV (Figura 4.1), projetada com foco na simplicidade e na tarefa primária do usuário: a busca por informações. Após a confirmação de que o usuário é um profissional da saúde, a busca pode ser feita tanto pelo nome do medicamento quanto pelo nome do princípio ativo, com o objetivo de abranger o maior número possível de resultados relevantes.

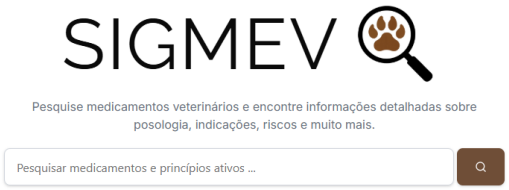


Figura 4.1: Tela inicial de consulta pública do SIGMEV.

Após a realização de uma busca, o sistema apresenta os resultados em uma lista clara e objetiva (Figura 4.2), onde cada item exibe informações essenciais para a identificação do fármaco.

Ao selecionar um dos resultados, o usuário é direcionado para a página de detalhamento do medicamento (Figura 4.3). A tela é estruturada para apresentar as informações mais críticas de forma imediata no topo da página, como princípio ativo e categorias. Os dados mais extensos, como posologia e restrições de uso, podem variar de acordo com a espécie, por isso, é possível navegar entre as espécies associadas a este medicamento para obter informações destes campos.


Figura 4.2: Tela com os resultados da pesquisa do princípio ativo amoxicilina.

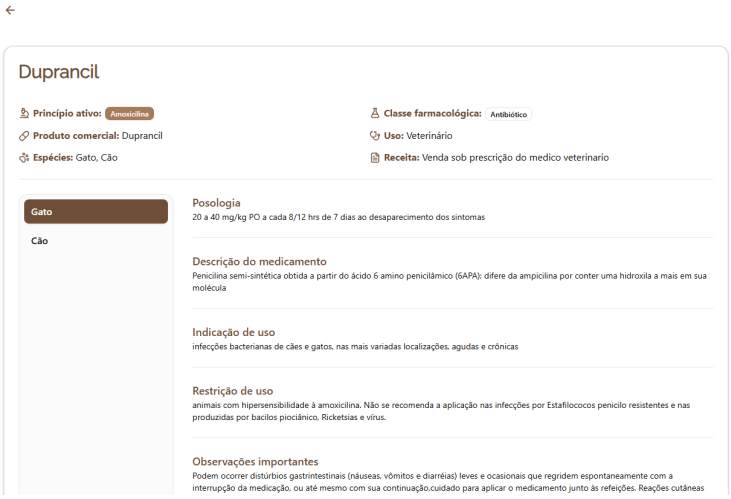


Figura 4.3: Página de detalhes do medicamento Duprancil.

### 4.3.2 Painel de Gerenciamento Administrativo

O acesso à área de gestão do sistema é protegido por uma tela de autenticação (Figura 4.4), garantindo que apenas usuários autorizados possam manipular os dados. Esse processo foi implementado utilizando o módulo *Auth* do *framework AdonisJS*, que fornece uma estrutura robusta para controle de sessão e verificação de credenciais.

Quando o usuário insere suas informações de *login*, é validado os dados inseridos com o que está cadastrado no banco de dados e, em caso de sucesso, cria-se uma sessão autenticada, permitindo o acesso às rotas e funcionalidades restritas. Caso as credenciais sejam inválidas, uma mensagem de erro é exibida, orientando o usuário

a corrigir as informações. Esse mecanismo assegura a integridade e a segurança do ambiente administrativo, impedindo acessos não autorizados e protegendo os dados sensíveis.

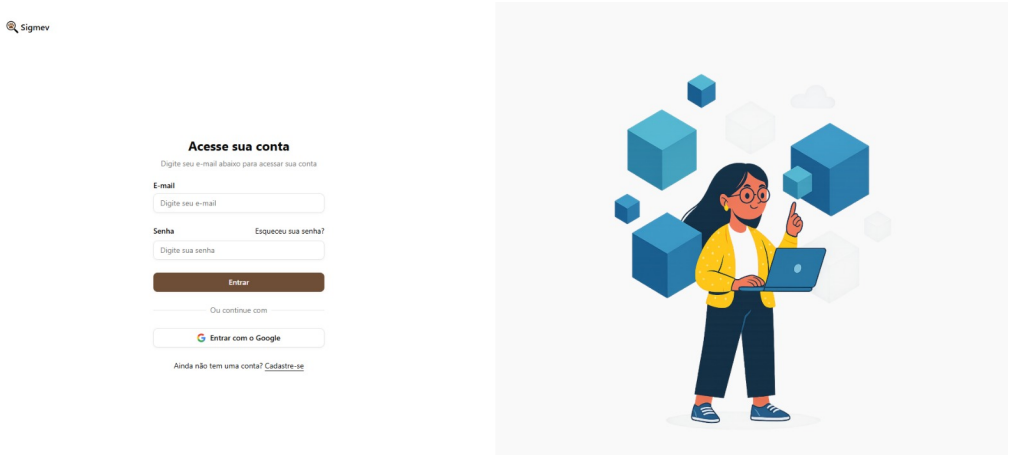


Figura 4.4: Tela de autenticação para acesso à área administrativa.

Após o *login*, o administrador é direcionado ao painel de gerenciamento de medicamentos (Figura 4.5). Essa área restrita pode ser acessada por Administradores, Profissionais da Saúde e Alunos, e sua visualização varia conforme o papel do usuário. Administradores têm acesso completo a todas as funcionalidades de gerenciamento, incluindo medicamentos e usuários. Profissionais da Saúde podem acessar apenas as ferramentas relacionadas à gestão de medicamentos. Já os Alunos podem visualizar a tabela de medicamentos e também cadastrar novos itens, mas sem acesso às demais funcionalidades administrativas.

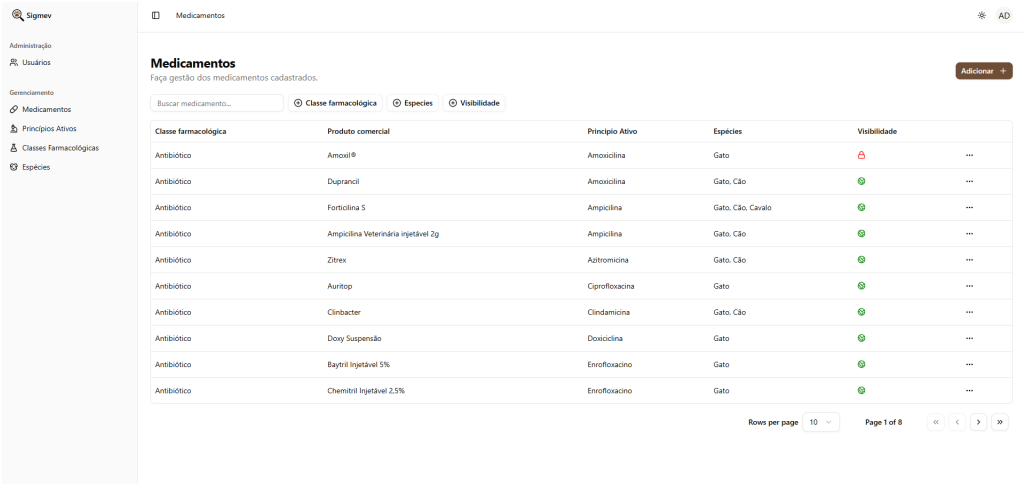


Figura 4.5: Painel administrativo na visão administrador.

### 4.3.3 Formulários de cadastro de medicamentos e edição

A inserção e a atualização dos dados, que representam o núcleo da gestão de conteúdo, ocorrem através de um formulário dedicado e consistente, utilizado tanto para adicionar (Figura 4.6) quanto para editar um medicamento (Figura 4.7). A consistência entre as duas operações foi uma decisão de *design* deliberada para reduzir a carga cognitiva do usuário, que interage com uma interface familiar independentemente da tarefa.

O formulário foi projetado para ser claro e intuitivo, com campos bem definidos que correspondem diretamente aos atributos do modelo de dados. Campos obrigatórios são marcados com um asterisco (*), uma prática de usabilidade que guia o usuário e atua como uma primeira camada de validação para garantir a integridade dos dados inseridos.

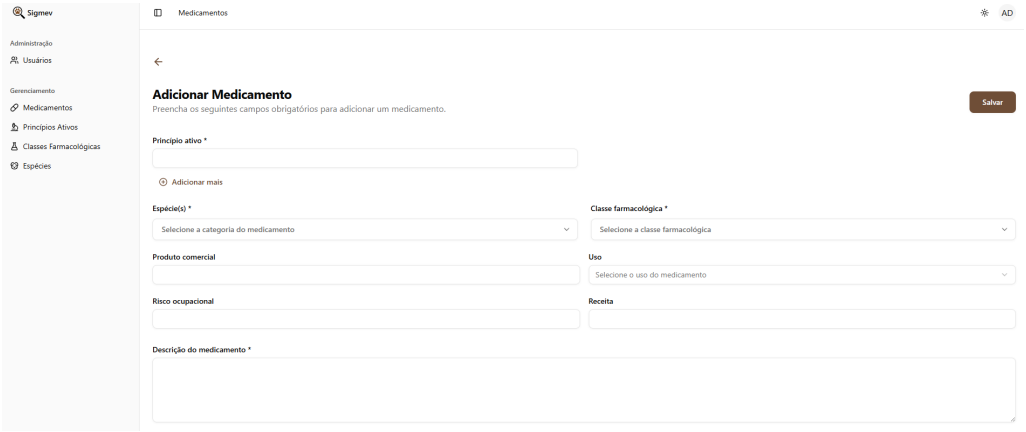


Figura 4.6: Formulário para o cadastro de um novo medicamento.

### 4.3.4 Painel de gerenciamento de usuários

Também o SIGMEV conta com um painel de gerenciamento de usuários, acessível exclusivamente por administradores (Figura 4.8), com o objetivo de assegurar o controle e a integridade dos dados cadastrados. Nesse painel, o administrador possui permissões avançadas, podendo cadastrar novos usuários e definir seus respectivos perfis de acesso (Figura 4.9).

Os papéis a serem cadastrados podem ser: administrador, profissional da saúde ou

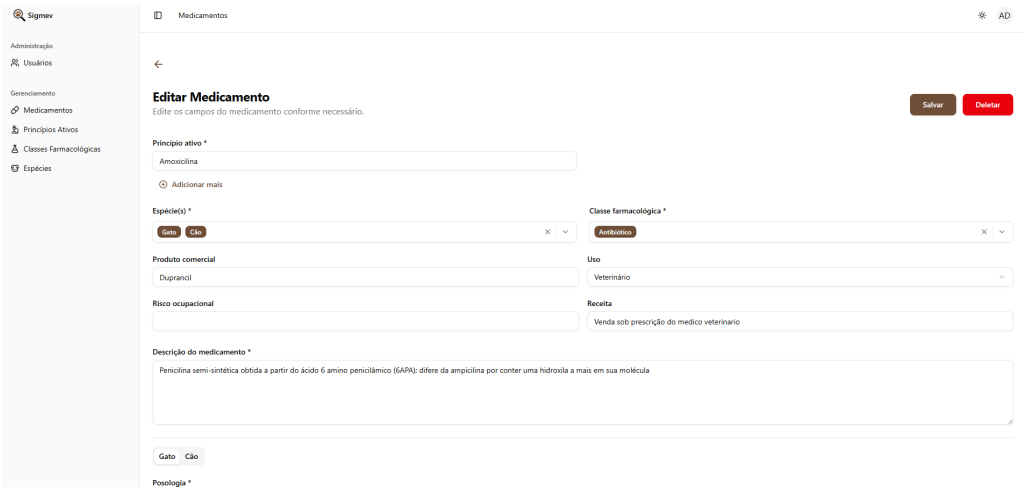


Figura 4.7: Formulário para a edição de um medicamento existente.

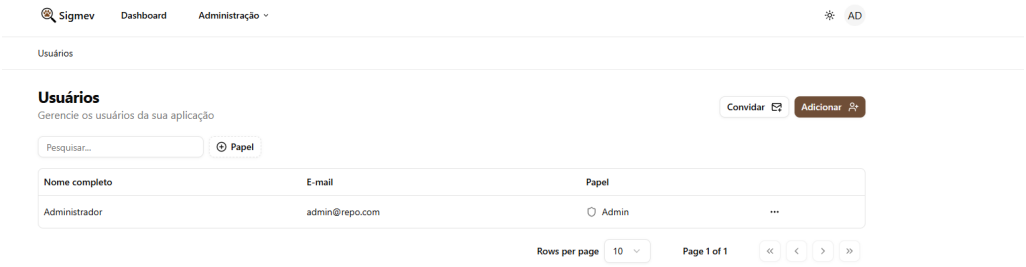


Figura 4.8: Painel de gerenciamento de usuários.

aluno. Além disso, é possível editar informações cadastrais (Figura 4.10) e remover usuários do sistema, garantindo que apenas pessoas autorizadas mantenham acesso ativo.

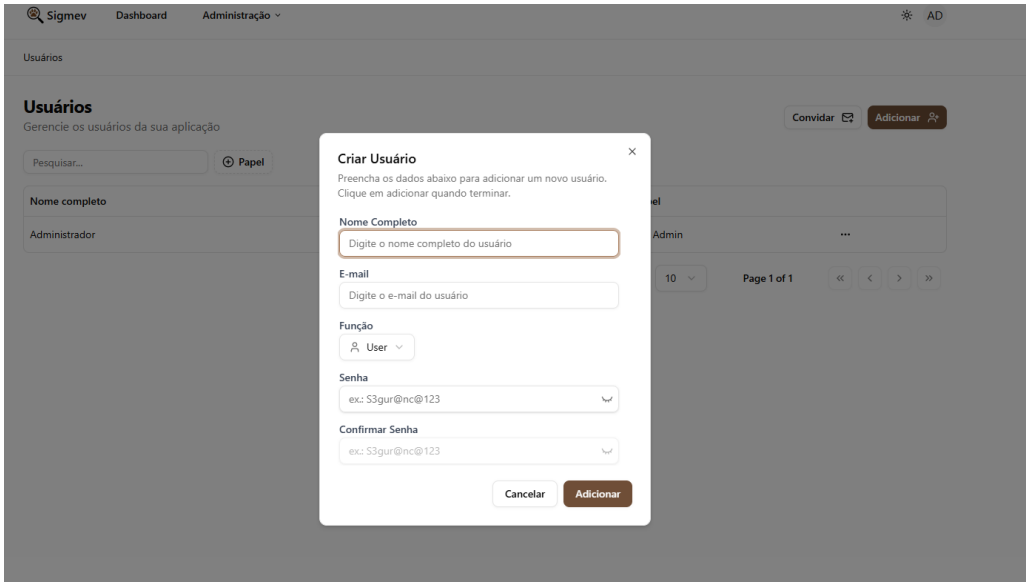


Figura 4.9: Formulário para o cadastro de um novo usuário.

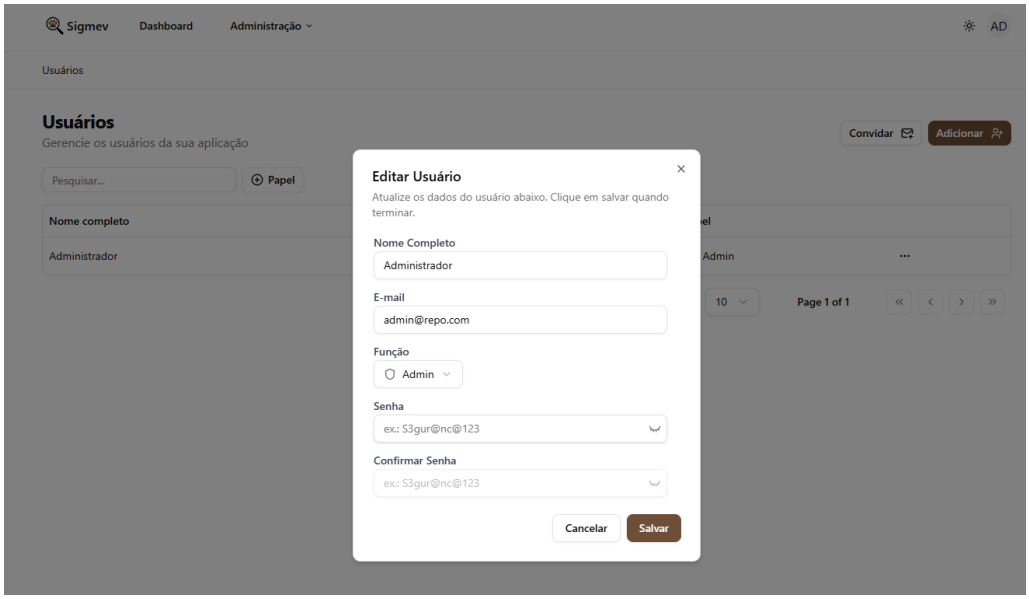


Figura 4.10: Formulário para edição de um usuário.

### 4.3.5 Outros gerenciamentos

Além do gerenciamento de medicamentos e usuários, o sistema oferece ao administrador e ao profissional da saúde usuários uma série de tabelas auxiliares que sustentam o correto preenchimento e classificação dos registros. A primeira delas é a tabela de Princípios Ativos (Figura 4.11), que reúne todas as substâncias utilizadas

como base farmacológica nos medicamentos cadastrados. Essa tabela permite manter um controle estruturado dos compostos disponíveis, permitindo a edição dos nomes de princípios ativos ou exclusão se este não estiver associado a nenhum medicamento.


Figura 4.11: Tabela de princípios ativos.

Em seguida, o sistema disponibiliza a tabela de Classes Farmacológicas (Figura 4.12), onde cada classe representa um agrupamento funcional dos medicamentos conforme sua categoria. É permitido ao usuário autorizado fazer a criação, edição ou exclusão de uma classe farmacológica.

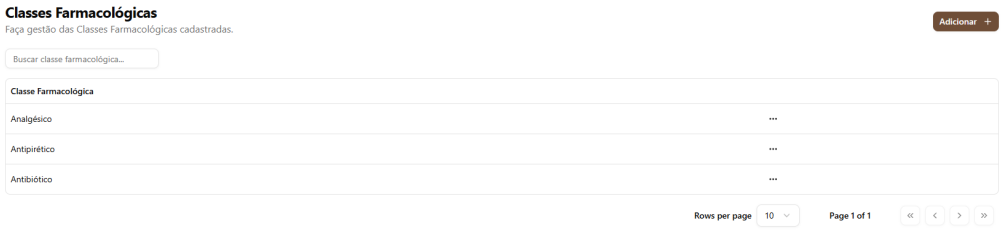


Figura 4.12: Tabela de classes farmacológicas.

Por fim, a tabela de Espécies (Figura 4.13) contém a lista das espécies animais contempladas pelo sistema. Ela é essencial para garantir que cada medicamento seja corretamente associado às espécies para as quais seu uso é indicado ou contraindicado, contribuindo para a precisão das decisões terapêuticas e para a segurança no tratamento veterinário.


Figura 4.13: Tabela espécies.

# Capítulo 5

# Conclusão

Este capítulo apresenta as considerações finais do trabalho, destacando as contribuições alcançadas com o desenvolvimento do sistema SIGMEV, bem como as principais limitações identificadas e propostas de aprimoramentos futuros. Este trabalho discutiu desde o contexto crítico da consulta a medicamentos de uso veterinário no Brasil até a construção de uma solução tecnológica capaz de reduzir algumas lacunas informacionais e apoiar profissionais, estudantes e pesquisadores da área.

## 5.1 Considerações finais

O presente trabalho teve como foco o desenvolvimento de um sistema web voltado ao cadastro, gerenciamento e consulta estruturada de medicamentos de uso veterinário. A motivação surgiu da constatação, apresentada no Capítulo 1, de que a fragmentação de informações técnicas, a falta de bases consolidadas e a dificuldade de acesso rápido e confiável a dados farmacológicos ainda representam obstáculos significativos na prática veterinária atual. O SIGMEV foi criado como uma resposta direta a essas limitações, buscando oferecer uma plataforma centralizada, intuitiva e segura para a consulta de informações essenciais à tomada de decisão clínica.

O desenvolvimento deste projeto foi baseado nos conceitos revisados no Capítulo 2, especialmente sobre o funcionamento do *React* e sua evolução. Entender princípios

como renderização declarativa, *Virtual DOM, Hooks* e os recursos mais recentes das versões 18 e 19 ajudou a estruturar um *front-end* responsivo e eficiente. Esses fundamentos foram importantes para tornar possível funcionalidades como filtragens em tempo real, carregamento dinâmico de componentes e uma experiência de uso fluida, características fundamentais para lidar com listas extensas e atualizações constantes de dados.

A proposta no Capítulo 3 e a implementação descrita no Capítulo 4 consolidou o SIGMEV como uma aplicação modular, escalável e adequada às necessidades identificadas. Foram desenvolvidos módulos completos para cadastro, edição, busca e visualização detalhada de medicamentos, além da aplicação de regras de negócio que garantem consistência e padronização das informações. Demonstrando ser capaz de organizar e apresentar dados essenciais, como princípios ativos, restrições de uso, classes farmacológicas e espécies indicadas, oferecendo uma consulta confiável e acessível.

Do ponto de vista prático, o SIGMEV representa uma contribuição relevante ao apoiar tanto atividades acadêmicas quanto rotinas da prática veterinária. A plataforma reduz problemas decorrentes de informações dispersas, agiliza a consulta e aumenta a segurança no uso de medicamentos. Sob o ponto de vista tecnológico, este trabalho mostra como ferramentas modernas de desenvolvimento web podem ser aplicadas de maneira eficaz para solucionar demandas específicas da área veterinária, unindo simplicidade de uso com solidez técnica.

Além disso, o estudo contribui para a discussão sobre como sistemas de informação podem melhorar a qualidade e a acessibilidade dos dados farmacológicos veterinários, uma área que ainda apresenta carências tanto na literatura quanto na oferta de soluções digitais especializadas. Dessa forma, o SIGMEV cumpre seu propósito ao auxiliar o acesso às informações sobre medicamentos de uso veterinário, proporcionando maior precisão, organização e rapidez.

## 5.2 Limitações e trabalhos futuros

Apesar dos avanços proporcionados pelo SIGMEV, algumas limitações ainda podem ser observadas na versão atual do sistema e merecem atenção em melhorias futuras. A principal delas está relacionada ao processo de inserção e atualização das informações. Hoje, o cadastro e a revisão dos dados dependem totalmente da intervenção manual dos responsáveis técnicos. Embora isso garanta maior controle sobre o conteúdo, também pode tornar o fluxo de atualização mais lento e sujeito a inconsistências, especialmente em cenários de grande volume de medicamentos ou quando ocorre a necessidade de revisões frequentes.

Outra limitação está na ausência de mecanismos automatizados de verificação, como validação cruzada com bases de dados oficiais, integração com APIs de órgãos reguladores ou ferramentas que ajudem a prevenir duplicidade, erros de preenchimento ou divergências terminológicas. A adoção desses recursos permitiria ampliar a precisão e a confiabilidade das informações, além de reduzir a carga operacional dos gestores do sistema.

Do ponto de vista funcional, embora o SIGMEV atenda aos requisitos principais estabelecidos, ainda há espaço para expandir suas capacidades. Uma das possibilidades para trabalhos futuros envolve a criação de um módulo avançado de relatórios e estatísticas, permitindo que o sistema também seja utilizado para fins de pesquisa, monitoramento institucional e apoio à atividade acadêmica. A ampliação do escopo de dados também é uma possibilidade, incluindo categorias como interações entre medicamentos, calculadora de doses, farmacologia avançada ou até alertas automáticos sobre restrições legais.

Uma outra etapa fundamental de trabalhos futuros é a validação do SIGMEV com usuários reais, envolvendo profissionais da saúde veterinária, docentes e discentes da área. Essa validação poderá ser conduzida por meio de estudos de uso controlados, aplicação de questionários de usabilidade, entrevistas semiestruturadas e análise de métricas objetivas, como tempo de execução de tarefas, taxa de erros e nível de satisfação dos usuários. O objetivo dessa avaliação é verificar a aderência do

sistema às necessidades práticas do público-alvo, identificar dificuldades de uso, lacunas funcionais e oportunidades de melhoria na interface e nos fluxos de interação. Os resultados obtidos a partir dessa validação empírica servirão como base para o refinamento contínuo do sistema, contribuindo para o aumento de sua eficiência, usabilidade e aceitação no contexto acadêmico e profissional.

Como umas das principais evoluções futuras, prevê-se também a integração do SIGMEV ao *Model Context Protocol* (MCP), um padrão recente para conexão entre sistemas e modelos de Inteligência Artificial, possibilitando que agentes automatizados consultem, processem e atualizem informações do banco de medicamentos de forma segura e auditável, ampliando significativamente o potencial de uso da plataforma.

# Referências


ABRAMOV, D. *React v.16.8: The One With Hooks*. 2019. Procure por "React v.16.8: The One With Hooks". Disponível em: <https://legacy.reactjs.org/blog/all.html/>.

ABRAMOV, D.; NABORS, R. *React v.17.0*. 2020. Procure por "React v.17.0". Disponível em: <https://legacy.reactjs.org/blog/all.html/>.

ACCOMAZZO, A.; MURRAY, N.; LERNER, A. *Fullstack React: The Complete Guide to ReactJS and Friends*. 2. ed. San Francisco: Fullstack.io, 2017.

Agência Nacional de Vigilância Sanitária (ANVISA). *Resíduos de medicamentos veterinários em alimentos de origem animal - RDC 328/2019*. 2019.

ANACLETO, T. A. et al. Erros de medicaç ao: Encarte de Farm'acia Hospitalar. *Revista Pharmacia Brasileira*, Conselho Federal de Farm'acia, n. 124, p. 1–12, 2010. Disponível em: <https://www.cff.org.br/sistemas/geral/revista/pdf/124/encarte_farmaciahospitalar.pdf>.

BANKS; PORCELLO. *Learning React: Modern Patterns for Developing React Apps*. 2. ed. [S.l.]: O'Reilly, 2020.

CLARK, A. *React v.16*. 2017. Procure por "React v.16". Disponível em: <https://legacy.reactjs.org/blog/all.html/>.

DATE, C. J. *An Introduction to Database Systems*. 8. ed. Boston: Addison-Wesley, 2004.

DODDS, K. C. *React Hooks: What's going to happen to my tests?* 2018. Disponível em: <https://kentcdodds.com/blog/react-hooks-whats-going-to-happen-to-my-tests>.

FEDOSEJEV, A. *React Essentials*. [S.l.]: Packt, 2015.

FLANAGAN, D. *Javascript: The Definitive Guide*. 7. ed. Sebastopol: O'Reilly Media, 2020.

GACKENHEIMER, C. *Introduction to React*. [S.l.]: Apress, 2015.

GAMA, T. A competência do farmacêutico na dispensação de medicamentos de uso veterinário: uma revisão bibliográfica. *Revista Foco*, v. 17, n. 11, p. 01–27, 2024.

GEEKSFORGEEKS. *React 19: New Features and Updates*. 2025. Disponível em: <https://www.geeksforgeeks.org/reactjs/react-19-new-features-and-updates/>.

KAPOOR, S. *React 18 New Features – Concurrent Rendering, Automatic Batching, and More*. 2022. Disponível em: <https://www.freecodecamp.org/news/react-18-new-features/>.

KRASNER, G. E.; POPE, S. T. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *Object-Oriented Programming*, v. 1, n. 3, p. 26–49, 1988.

LUSTGARTEN, J. L. et al. Veterinary informatics: forging the future between veterinary medicine, human medicine, and One Health initiatives. *JAMIA Open*, Oxford University Press, v. 3, n. 2, p. 306–317, 2020.

Ministério da Agricultura, Pecuária e Abastecimento. *Portaria SDA/MAPA nº 200, de 22 de janeiro de 2021*. 2021. Diário Oficial da União. Estabelece procedimentos para adequação de registros de produtos veterinários conforme limites máximos de resíduos definidos pela Anvisa.

Ministério da Saúde. *Uso Racional de Medicamentos*. 2024. Disponível em: <https://www.gov.br/saude/pt-br/composicao/sectics/daf/uso-racional-de-medicamentos>.

MINNICK, C. *Beginning ReactJS Foundations Building User Interfaces with ReactJS*. 1. ed. [S.l.]: Wiley, 2022.

NALAWADE, K. *React 19: New Hooks Explained with Examples*. 2024. Disponível em: <https://www.freecodecamp.org/news/react-19-new-hooks-explained-with-examples/>.

NOVOTNY, M. *whats-new-in-react-19*. 2024. Disponível em: <https://vercel.com/blog/whats-new-in-react-19>.

PAESE, K.; JESUS, L. D.; ANDRADE, D. D. Aspectos regulatórios na produção de medicamentos veterinários em escala magistral e industrial. *Infarma – Ciências Farmacêuticas*, v. 35, n. 1, p. 14–28, 2023.

PINHO, R. H. P.; NASR-ESFAHANI, M.; PANG, D. S. J. Medication errors in veterinary anesthesia: a literature review. *Veterinary Anaesthesia and Analgesia*, v. 51, n. 3, p. 203–226, 2024.

PLUMB´S. *Plumb's Veterinary Drug Handbook Online*. 2025. Disponível em: <https://plumbs.com/>.

POP, D.-P.; ALTAR, A. Designing an mvc model for rapid web application development. *24th DAAAM International Symposium on Intelligent Manufacturing and Automation*, 2013.

POPA, A.; ALBERT, P. A comparative study of javascript frameworks. *Procedia Computer Science*, v. 141, n. 3, p. 97–104, 2018.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 7. ed. New York: McGraw-Hill, 2010.

RAFAL, B. *Concurrent rendering. Let's take a closer look at React 18 concurrent mode*. 2021. Disponível em: <https://tsh.io/blog/react-concurent-rendering>.

SANTAMARIA, S. L.; ZIMMERMAN, K. L. Uses of informatics to solve real world problems in veterinary medicine. *Journal of Veterinary Medical Education*, University of Toronto Press, v. 38, n. 2, p. 103–109, 2011.

SHAKIKHANLI, U.; BILICKI, V. Optimizing branching strategies in mono-and multi-repository environments: A comprehensive analysis. *Computer Assisted Methods in Engineering and Science*, v. 31, n. 1, p. 81–111, 2024.

SOMMERVILLE, I. *Software Engineering*. 10. ed. Boston: Pearson, 2016.

SRINIVASAN, V. et al. Platfab: A platform engineering approach to improve developer productivity. *Journal of Information Systems Engineering and Business Intelligence*, v. 11, n. 1, 2025.

VLASIOU, M. C. Vet informatics and the future of drug discovery in veterinary medicine. *Frontiers in Veterinary Science*, v. 11, 2024. Opinion article, Sec. Veterinary Pharmacology and Toxicology. Disponível em: <https://www.frontiersin.org/journals/veterinary-science/articles/10.3389/fvets.2024.1494242>.

WIERUCH, R. *Road to React*. 4. ed. [S.l.]: LeanPub, 2022.

ZAKAS, N. C. *Understanding ECMAScript 6: The Definitive Guide for JavaScript Developers*. 1. ed. San Francisco: No Starch Press, 2016.

# Apêndice A

# Especificação de Casos de Uso

- **CU-01: Cadastrar Usuário**

  **Ator:** Administrador.

  **Descrição:** O administrador pode cadastrar usuários definindo seus papéis (Administrador, Profissional da Saúde ou Aluno, conforme **RN-01**).

  **Pré-condição:** O administrador deve estar autenticado no sistema (Conforme **RN-03**).

  **Fluxo principal:**

  1. O administrador acessa o módulo de gerenciamento de usuários.
  2. O sistema exibe a lista de usuários cadastrados.
  3. O administrador solicita adicionar um novo usuário.
  4. O sistema exibe o formulário e solicita os dados a serem preenchidos (conforme **RN-01**).
  5. O administrador preenche os dados solicitados.
  6. O sistema valida os dados e confirma a operação.
  7. Fim do caso de uso.

  **Fluxos de Exceção:**

  **6.a:** Campos obrigatórios não preenchidos.

1. O sistema impede a confirmação e exibe uma mensagem de erro detalhando os campos pendentes.
2. Retorna ao passo 4 do Fluxo Principal.

**6.b:** E-mail de usuário já existente.

1. O sistema rejeita a operação e notifica ao administrador sobre a duplicidade.
2. Retorna ao passo 4 do Fluxo Principal.

**Pós-condição:** Os dados do usuário são cadastrados com sucesso.

- **CU-02: Editar Usuário**

  **Ator:** Administrador.

  **Descrição:** O administrador pode editar usuários (conforme **RN-01**).

  **Pré-condição:** O administrador deve estar autenticado no sistema (Conforme **RN-03**).

  **Fluxo principal:**

  1. O administrador acessa o módulo de gerenciamento de usuários.
  2. O sistema exibe a lista de usuários cadastrados.
  3. O administrador solicita a edição de um usuário.
  4. O sistema exibe os dados do usuário (conforme **RN-01**).
  5. O administrador altera os dados pendentes e confirma.
  6. O sistema valida os dados e confirma a operação.
  7. Fim do caso de uso.

  **Fluxos de Exceção:**

  **6.a:** Campos obrigatórios não preenchidos.

  1. O sistema impede a confirmação e exibe uma mensagem de erro detalhando os campos pendentes.
  2. Retorna ao passo 4 do Fluxo Principal.

**6.b:** E-mail de usuário já existente.

1. O sistema rejeita a operação e notifica ao administrador sobre a duplicidade.
2. Retorna ao passo 4 do Fluxo Principal.

**Pós-condição:** Os dados do usuário são editados com sucesso.

- **CU-03: Excluir Usuário**

  **Ator:** Administrador.

  **Descrição:** O administrador pode excluir usuários (conforme **RN-02**).

  **Pré-condição:** O administrador deve estar autenticado no sistema (Conforme **RN-03**).

  **Fluxo principal:**

  1. O administrador acessa o módulo de gerenciamento de usuários.
  2. O sistema exibe a lista de usuários cadastrados.
  3. O administrador solicita a exclusão de um usuário.
  4. O sistema solicita confirmação explícita e preenchimento do e-mail do usuário a ser excluído (conforme **RN-02**).
  5. O administrador informa o e-mail e confirma.
  6. O sistema remove o usuário.
  7. Fim do caso de uso.

  **Fluxos de Exceção:**

  **5.a:** O administrador cancela a exclusão.

  1. O administrador solicita o cancelamento da operação.
  2. Retorna ao passo 2 do Fluxo Principal.

  **Pós-condição:** Os dados do usuário são removidos com sucesso.

- **CU-04: Cadastrar Medicamento**

  **Ator:** Usuário (Generalização de Administrador, Profissional da Saúde e

Aluno).

**Descrição:** Permitir o cadastro completo de medicamentos de uso veterinário (conforme **RN-04**).

**Pré-condição:** O usuário deve estar autenticado.

**Fluxo principal:**

1. O usuário acessa o formulário de cadastro de medicamentos.
2. O sistema exibe os campos a serem preenchidos (conforme **RN-04**).
3. O usuário preenche os dados e confirma o envio.
4. O sistema valida os dados, armazena as informações com visibilidade Privada (conforme **RN-05**) e notifica o usuário sobre o sucesso da operação.
5. Fim do caso de uso.

**Fluxos de Exceção:**

**4.a:** Campos obrigatórios não preenchidos.

1. O sistema impede o armazenamento e destaca os campos faltantes com uma mensagem de erro.
2. Retorna ao passo 2 do Fluxo Principal.

**Pós-condição:** O medicamento é adicionado à base de dados com visibilidade privada (conforme **RN-05**).

- **CU-05: Editar Medicamento**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir a edição das informações de um medicamento, incluindo a alteração de sua visibilidade (conforme **RN-04** e **RN-05**).

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo de gerenciamento de medicamentos.
  2. O sistema exibe a lista de medicamentos.

3. O usuário seleciona um medicamento.
4. O sistema exibe os campos com os dados preenchidos (conforme **RN-04**).
5. O usuário altera os dados desejados e confirma as alterações.
6. O sistema valida e salva os novos dados.
7. Fim do caso de uso.

**Pós-condição:** Os dados do medicamento são atualizados com sucesso.

- **CU-06: Excluir Medicamento**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir a exclusão de um medicamento cadastrado (conforme **RN-06**).

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo de gerenciamento de medicamentos.
  2. O sistema exibe a lista de medicamentos.
  3. O usuário solicita a exclusão do medicamento.
  4. O sistema solicita confirmação explícita (conforme **RN-06**).
  5. O usuário confirma a exclusão.
  6. O sistema remove o medicamento da base de dados.
  7. Fim do caso de uso.

  **5.a:** Exclusão cancelada.

  1. O usuário solicita o cancelamento da operação.
  2. Retorna ao passo 2 do Fluxo Principal.

  **Pós-condição:** O medicamento é removido permanentemente do sistema.

- **CU-07: Buscar e Consultar Medicamento (Público)**

  **Ator:** Usuário (Generalização de Público geral).

**Descrição:** Permitir que o usuário não autenticado busque medicamentos (conforme **RN-08**) e consulte seus detalhes, mediante a declaração de que é um profissional da saúde (conforme **RN-07**).

**Pré-condição:**

**Fluxo principal:**

1. O usuário acessa a página de busca pública.
2. O sistema exibe um aviso solicitando a declaração "Sou um profissional da saúde" (**RN-07**).
3. O usuário aceita a declaração.
4. O sistema habilita a barra de busca.
5. O usuário insere termos de busca (nome comercial ou princípio ativo, conforme **RN-08**).
6. O sistema exibe a lista de medicamentos com visibilidade "Pública" que correspondem à busca.
7. O usuário seleciona um medicamento para consultar seus detalhes.
8. Fim do caso de uso.

**Fluxos de Exceção:**

**3.a:** Usuário não aceita a declaração.

1. O sistema impede a habilitação da busca (passo 4) e informa que o acesso é restrito.
2. Fim do caso de uso.

**6.a:** Nenhum resultado encontrado.

1. O sistema exibe a mensagem "Nenhum medicamento ou princípio ativo encontrado" e mantém o usuário na tela de busca (passo 4).
2. Fim do caso de uso.

**Pós-condição:** O usuário consulta os detalhes dos medicamentos públicos.

- **CU-08: Editar Princípios Ativos**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir a edição de princípios ativos (conforme **RN-09**). O cadastro é realizado via **CU-05**.

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo "Princípios Ativos".
  2. O sistema exibe a lista de princípios ativos cadastrados.
  3. O usuário solicita a edição de um princípio ativo.
  4. O sistema exibe os dados e permite a alteração do nome (conforme **RN-09**).
  5. O usuário edita e confirma.
  6. O sistema valida e salva a alteração.
  7. Fim do caso de uso.

  **Pós-condição:** O princípio ativo é editado.

- **CU-09: Excluir Princípios Ativos**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir a exclusão de princípios ativos (conforme **RN-10**).

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo "Princípios Ativos".
  2. O sistema exibe a lista de princípios ativos cadastrados.
  3. O usuário solicita a exclusão de um princípio ativo.
  4. O sistema solicita confirmação.
  5. O usuário confirma.

6. O sistema valida e remove o princípio ativo.
7. Fim do caso de uso

**Fluxos de Exceção:**

**6.a:** Exclusão negada (princípio ativo em uso).

1. O sistema verifica se o princípio ativo está relacionado a um medicamento (conforme RN-**10**).
2. O sistema exibe a mensagem "Este item não pode ser excluído pois está associado a um medicamento".
3. Retorna ao passo 2 do fluxo principal.

**Pós-condição:** O princípio ativo é removido.

- **CU-10: Cadastrar Classe Farmacológica**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir o cadastro de classes farmacológicas (conforme RN-**11**).

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme RN-**03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo "Classes farmacológicas".
  2. O sistema exibe a lista de classes cadastradas.
  3. O usuário solicita o cadastro da classe farmacológica.
  4. O sistema exibe os dados a serem preenchidos (conforme RN-**11**).
  5. O usuário preenche o campo e confirma.
  6. O sistema salva a nova classe e atualiza a lista.
  7. Fim do caso de uso.

  **Pós-condição:** Uma nova classe farmacológica é cadastrada.

- **CU-11: Editar Classe Farmacológica**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

**Descrição:** Permitir a edição de Classes Farmacológicas (conforme **RN-11**).

**Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

**Fluxo principal:**

1. O usuário acessa o módulo "Classes farmacológicas".
2. O usuário solicita a edição da classe farmacológica.
3. O sistema exibe os dados a serem preenchidos (conforme **RN-11**).
4. O usuário edita o campo e confirma.
5. O sistema salva a edição da classe e atualiza a lista.
6. Fim do caso de uso.

**Pós-condição:** A classe farmacológica é editada.

- **CU-12: Excluir Classe Farmacológica**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir a exclusão de classes farmacológicas (conforme **RN-12**).

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo "Classes farmacológicas".
  2. O sistema exibe a lista de classes cadastradas.
  3. O usuário seleciona a exclusão da classe farmacológica.
  4. O sistema solicita confirmação.
  5. O usuário confirma.
  6. O sistema valida e remove a classe.
  7. Fim do caso de uso.

  **Fluxos de Exceção:**

  **6.a:** Exclusão negada (classe em uso).

1. O sistema verifica se a classe farmacológica está relacionado a um medicamento (conforme **RN-12**).
2. O sistema exibe a mensagem “Este item não pode ser excluído pois está associado a um medicamento”.
3. Retorna ao passo 2 do fluxo principal.

**Pós-condição:** A classe farmacológica é excluída.

- **CU-13: Cadastrar Espécies Animais**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir o cadastro de espécies animais (conforme **RN-13**).

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo “Espécies”.
  2. O sistema exibe a lista de espécies cadastradas.
  3. O usuário solicita o cadastro da espécie.
  4. O sistema exibe os dados a serem preenchidos (conforme **RN-13**).
  5. O usuário preenche o campo e confirma.
  6. O sistema salva a nova espécie e atualiza a lista.
  7. Fim do caso de uso.

  **Pós-condição:** Uma nova espécie é cadastrada.

- **CU-14: Editar Espécies Animais**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir a edição de espécies animais (conforme **RN-13**).

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo “Espécies”.

2. O sistema exibe a lista de espécies cadastradas.
3. O usuário solicita a edição da espécie.
4. O sistema exibe os dados a serem editados (conforme **RN-13**).
5. O usuário preenche o campo e confirma.
6. O sistema salva a edição da espécie e atualiza a lista.
7. Fim do caso de uso.

**Pós-condição:** A espécie é editada com sucesso.

- **CU-15: Excluir Espécies Animais**

  **Ator:** Usuário (Generalização de Administrador e Profissional da Saúde).

  **Descrição:** Permitir a exclusão de espécies animais (conforme **RN-14**).

  **Pré-condição:** O usuário deve estar autenticado e possuir permissão (conforme **RN-03**).

  **Fluxo principal:**

  1. O usuário acessa o módulo "Espécies".
  2. O sistema exibe a lista de espécies cadastradas.
  3. O usuário solicita a exclusão da espécie.
  4. O sistema solicita confirmação.
  5. O usuário confirma.
  6. O sistema valida e remove a espécie.
  7. Fim do caso de uso.

  **Fluxos de Exceção:**

  **6.a:** Exclusão negada (espécie em uso).

  1. O sistema verifica se a espécie está relacionado a um medicamento (conforme **RN-14**).
  2. O sistema exibe a mensagem "Este item não pode ser excluído pois está associado a um medicamento".

3. Retorna ao passo 2 do fluxo principal.

**Pós-condição:** A espécie é excluída.