CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE LINS PROF. ANTONIO SEABRA
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ERIC HENRIQUE SILVA RICOLDI

PORTFÓLIO ACADÊMICO

LINS/SP
1º SEMESTRE/2026

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE LINS PROF. ANTONIO SEABRA
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ERIC HENRIQUE SILVA RICOLDI

PORTFÓLIO ACADÊMICO

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Lins Prof. Antônio Seabra, para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas

Orientador: Prof. Dr. Thiago Seti Patricio.

LINS/SP
1º SEMESTRE/2026

Ricoldi, Eric Henrique Silva

R541p Portfólio acadêmico / Eric Henrique Silva Ricoldi. — Lins, 2026.

102f.

Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) — Faculdade de Tecnologia de Lins Professor Antonio Seabra: Lins, 2026.

Orientador(a): Dr. Thiago Seti Patricio

1. Portfólio. 2. Projetos. 3. Desenvolvimento. 4. Conhecimento técnico e teórico. 5. Site. I. Patricio, Thiago Seti. II. Faculdade de Tecnologia de Lins Professor Antonio Seabra. III. Título.

CDD 004.21

Gerada automaticamente pelo módulo web de ficha catalografica da FATEC Lins mediante dados fornecidos pelo(a) autor(a).

ERIC HENRIQUE SILVA RICOLDI

PORTFÓLIO ACADÊMICO

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Lins Prof. Antônio Seabra, como parte dos requisitos para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas sob orientação do Prof. Dr. Thiago Seti Patricio.

Data de aprovação: ____/____/____

_____

Nome do Orientador

_____

Nome do Examinador 1

_____

Nome do Examinador 2

DEDICATÓRIA

Ao meu querido e já saudoso avô Osvailton Ricoldi. Jamais me esquecerei do amor e cuidado que ele tinha por mim. Foi fundamental não apenas para essa minha vitória, mas também, foi fundamental para a minha vida.

Eric Henrique Silva Ricoldi

AGRADECIMENTOS

Sou grato a Deus por minha vida e por ter me dado condições de concluir este trabalho de graduação. Sou grato à minha família pelo cuidado e apoio durante toda a trajetória do curso.

Sou grato ao meu orientador por ter dedicado tempo e esforço para ajudar-me neste trabalho. Sou grato aos professores que tive durante o curso. Admiro a competência de cada um em relação ao que ensinam e admiro a capacidade que eles têm de inspirar os alunos.

Sou grato aos meus colegas de turma que sempre me ajudaram nos momentos de dificuldade em algum trabalho ou disciplina. Também sou grato pelos momentos de conversa, risos, parceria e união.

Eric Henrique Silva Ricoldi

RESUMO

O portfólio acadêmico consiste na demonstração dos projetos mais significativos de cada semestre do curso de Análise e Desenvolvimento de Sistemas (ADS). O objetivo principal é demonstrar os conhecimentos adquiridos durante o curso de forma contextualizada e organizada. Para que o trabalho seja analisado por futuros alunos e empregadores não apenas tecnicamente, mas também pedagogicamente. Cada capítulo do trabalho apresenta um projeto desenvolvido em um semestre específico do curso. Compreendendo o período do 2º ao 6º semestre em ordem cronológica. Cada projeto aborda os seguintes tópicos: contextualização da disciplina e do objetivo do trabalho, conceitos teóricos relevantes, tecnologias e ferramentas utilizadas, explicação do código e uma breve conclusão. Como complemento à monografia foi elaborado o site do portfólio acadêmico, no qual se apresentam de forma sucinta e dinâmica os projetos presentes neste trabalho de graduação (TG). Mediante os projetos desenvolvidos durante a graduação, o discente, após a conclusão deste trabalho, pôde perceber sua evolução em relação ao conhecimento técnico e teórico, bem como os aspectos a serem aprimorados. O trabalho, portanto, foi benéfico não apenas ao formando, mas também buscou contribuir na divulgação do curso à comunidade e aos futuros alunos por meio da apresentação dos projetos disciplinares do curso.

Palavras-chave: Portfólio. Projetos. Desenvolvimento. Semestre. Site. Conhecimento técnico e teórico. Pedagógico.

ABSTRACT

The academic portfolio consists of a demonstration of the most significant projects from each semester of the Systems Analysis and Development course. The primary objective is to demonstrate the knowledge acquired during the course in a contextualized and organized manner, allowing the work to be analyzed not only technically but also pedagogically by future students and employers. Each chapter of the work presents a project developed in a specific semester of the course, covering the period from the 2nd to the 6th semester in chronological order. Each project addresses the following topics: contextualization of the discipline and the objective of the work, relevant theoretical concepts, technologies and tools used, code explanation, and a brief conclusion. As a complement to the monograph, the academic portfolio website was created, which displays the projects present in this graduation work in a concise and dynamic way. Through the projects developed during the graduation, the student, after completing this work, was able to perceive their evolution in relation to technical and theoretical knowledge, as well as the aspects to be improved. The work, therefore, was beneficial not only to the graduate but also sought to contribute to the dissemination of the course to the community and future students through the presentation of the course's disciplinary projects.

Keywords: Portfolio. Projects. Development. Semester. Site. Technical and theoretical knowledge. Pedagogical.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ADS - Análise e Desenvolvimento de Sistemas
API - *Application Programming Interface*
CSS - *Cascading StyleSheets*
CRUD - *Create* (Criar), *Read* (Ler), *Update* (Atualizar) e *Delete* (Excluir)
DTO - *Data Transfer Object*
FK - *Foreign Key*
HTML - *Hypertext Markup Language*
HTTP - *Hypertext Transfer Protocol*
IDE - *Integrated Development Environment*
IHC - Interação Humano-Computador
JSON - *JavaScript Object Notation*
JS - JavaScript
MVC - *Model-View-Controller*
MER - Modelo Entidade-Relacionamento
PK - *Primary Key*
PHP - *Hypertext Preprocessor*
RAM - *Random Access Memory*
SQL - *Structured Query Language*
SGBD - Sistema de Gerenciamento de Banco de Dados
URL - *Uniform Resource Locator*
UX - *User Experience*
XML - *Extensible Markup Language*
XSS - *Cross-Site Scripting*

LISTA DE SÍMBOLOS

*	-	Asterisco
<	-	Menor
>	-	Maior
#	-	Cerquilha
%	-	Porcentagem
&	-	E comercial
=	-	Igualdade
( )	-	Parênteses
@	-	Arroba
[ ]	-	Colchetes
$	-	Cifrão

SUMÁRIO

1 INTRODUÇÃO

Na era da digitalização, quase todas as atividades humanas são mediadas por sistemas computacionais. A Análise e Desenvolvimento de Sistemas (ADS) se apresenta como um campo crucial no mundo contemporâneo, pois, é responsável por planejar, implementar e gerenciar sistemas que automatizam processos e resolvem problemas em diferentes esferas da sociedade (Lucchesi, 2025).

A área de ADS vai muito além da escrita de códigos ou da construção de aplicações. Ela envolve a capacidade de compreender e solucionar problemas reais por meio da tecnologia, promovendo melhorias significativas em diversos contextos sociais e organizacionais. Além disso, o desenvolvimento de sistemas escaláveis e adaptáveis permite antecipar necessidades futuras, contribuindo para a inovação e para a evolução contínua dos processos tecnológicos. O analista é o profissional responsável por identificar problemas, levantar requisitos e propor soluções tecnológicas. Traduz as necessidades do cliente em especificações que podem ser transformadas em soluções práticas. O papel do desenvolvedor é transformar essas especificações em código. Ele domina linguagens de programação, frameworks e ferramentas de desenvolvimento. Atua, também, na realização de testes garantindo o funcionamento do sistema (Lucchesi, 2025).

Nesse contexto, Zenaro (2025), explica que o profissional analista e desenvolvedor de sistemas deve ser capaz de compreender desde o planejamento do software até seu desenvolvimento, combinando habilidades analíticas de planejamento e técnicas de programação.

Portanto, é essencial que os estudantes do curso de ADS desenvolvam um conjunto sólido de competências técnicas e cognitivas que lhe permitam interpretar problemas, propor soluções e acompanhar o ciclo completo de desenvolvimento de software. Tais competências são adquiridas de forma gradual, por meio de experiências práticas, projetos interdisciplinares e trabalhos aplicados ao longo do curso. Assim, o aprendizado deixa de ser apenas teórico e passa a estar diretamente vinculado a situações reais de desenvolvimento, o que contribui para a formação de um profissional mais reflexivo, autônomo e preparado para os desafios do mercado de trabalho contemporâneo.

Com o objetivo de sistematizar e demonstrar o conhecimento adquirido durante o percurso acadêmico, optou-se pela elaboração de um portfólio acadêmico como

trabalho de conclusão de curso (TCC). Esse formato foi escolhido por sua capacidade de reunir, em um único documento, as produções mais significativas realizadas ao longo da graduação. Por adendo, o portfólio permite evidenciar o crescimento técnico e intelectual do estudante, bem como sua evolução em termos de compreensão conceitual e capacidade de resolução de problemas. Além disso, ele serve como instrumento de autoavaliação, permitindo que o próprio estudante perceba sua trajetória de aprendizado e identifique seus pontos fortes e aspectos que podem ser aprimorados.

A adoção do modelo de portfólio como trabalho de graduação apresenta diversas vantagens. Em primeiro lugar, ele possibilita ao egresso demonstrar, de maneira organizada e contextualizada, as habilidades e conhecimentos que adquiriu em diferentes disciplinas e projetos. Em segundo lugar, constitui um importante diferencial competitivo no mercado de trabalho, podendo ser apresentado em processos seletivos, entrevistas ou eventos acadêmicos. O portfólio funciona, portanto, como uma vitrine profissional, que evidencia as habilidades do futuro analista. Além disso, serve de referência para futuros alunos e pesquisadores, que podem compreender o percurso formativo do curso a partir dos projetos descritos.

Cada capítulo deste trabalho apresenta um projeto desenvolvido em um semestre específico do curso, seguindo uma ordem cronológica que compreende o período do 2º ao 6º semestre letivo. Os projetos foram escolhidos com base em sua relevância acadêmica e complexidade técnica. Essa organização permite observar a progressiva ampliação do repertório do estudante, tanto em termos conceituais quanto em relação às tecnologias empregadas. O segundo capítulo, por exemplo, aborda o projeto realizado durante o 2º semestre, na disciplina de Interação Humano-Computador (IHC). Nessa atividade, foi elaborado o design de um formulário de inscrição online para um evento de conferência de tecnologia. O trabalho envolveu a criação de uma interface intuitiva e responsiva, com validação em tempo real dos campos do formulário e um alerta simulando o envio das informações. As tecnologias utilizadas foram *Hypertext Markup Language* (HTML), *Cascading StyleSheets* (CSS), JavaScript e Bootstrap. O projeto explorou aspectos da qualidade de software e experiência do usuário em todo o processo de desenvolvimento.

O terceiro capítulo apresenta o projeto desenvolvido no 3º semestre, na disciplina de Estrutura de Dados. Nessa etapa, foi implementado um sistema de gerenciamento para uma livraria, utilizando a linguagem de programação C. O sistema

foi implementado com o uso de *structs* e armazenamento de informações em arquivos binários, proporcionando uma experiência prática de manipulação de dados e persistência de informações em um contexto realista. Esse projeto consolidou conhecimentos fundamentais sobre estruturas de dados, lógica de programação e organização da memória computacional, competências indispensáveis para qualquer desenvolvedor de software.

O quarto capítulo descreve o projeto realizado no 4º semestre, na disciplina de Programação para Web. Desenvolveu-se uma *Application Programming Interface* (API) voltada à gestão de castrações de animais. O sistema foi implementado utilizando a linguagem Java, em conjunto com os *frameworks* Spring e Hibernate, o que permitiu o estudo de conceitos mais avançados, como a arquitetura *Model-View-Controller* (MVC), o mapeamento objeto-relacional e a integração entre camadas de aplicação. Ao estruturar o sistema sob uma arquitetura MVC foi possível aplicar modularidade e reaproveitamento de código no projeto. Os *frameworks* utilizados aceleraram e simplificaram configurações no desenvolvimento da API.

O quinto capítulo apresenta o projeto realizado no 5º semestre, na disciplina de laboratório de banco de dados. Esse projeto é uma demonstração de comandos *Structured Query Language* (SQL) de consulta em um banco de dados, utilizando o Sistema de Gerenciamento de Banco de Dados (SGBD) da oracle: SQL Developer. Esse trabalho evidenciou a importância geral da linguagem SQL para os desenvolvedores, analistas e outras profissões.

O sexto capítulo expõe o projeto realizado no 6º semestre, na disciplina de tópicos especiais em informática. Nele é descrito o desenvolvimento de uma aplicação *web* em PHP que possui funções de criação, leitura, atualização e exclusão de informações. Conta também com as funcionalidades de busca e paginação das informações. Esse trabalho proporcionou uma base de conhecimentos importantes para o desenvolvimento de aplicações *web* em PHP.

Para cada um dos projetos apresentados neste portfólio, serão abordados os seguintes tópicos: contextualização da disciplina e objetivo do trabalho, conceitos teóricos relevantes, tecnologias e ferramentas utilizadas, explicação de código e uma breve conclusão. Essa estrutura busca garantir uma leitura organizada e compreensível, de modo que cada projeto possa ser analisado tanto em sua dimensão técnica quanto pedagógica. Além disso, essa sistematização permite destacar a

coerência entre as competências desenvolvidas em diferentes disciplinas, revelando como o curso contribui de forma integrada para a formação profissional.

Em síntese, este trabalho tem como propósito evidenciar a trajetória formativa do estudante de DS, destacando as habilidades técnicas, metodológicas e reflexivas construídas ao longo do curso. Ao reunir projetos representativos de cada semestre, o portfólio proporciona uma visão panorâmica do aprendizado, demonstrando não apenas o domínio de linguagens e *frameworks*, mas também a capacidade de planejar, executar e documentar soluções tecnológicas alinhadas às demandas atuais do mercado. Dessa forma, o trabalho se configura como um importante registro acadêmico e profissional, reafirmando a relevância da prática integrada ao ensino como caminho para a formação de analistas e desenvolvedores capazes de atuar de forma crítica, criativa e ética no contexto da sociedade da informação.

2 PROJETO DO SEGUNDO SEMESTRE: FORMULÁRIO WEB DE INSCRIÇÃO

2.1 CONTEXTUALIZAÇÃO E OBJETIVO

Na disciplina de Interação Humano-Computador (IHC), lecionada pelo professor Diego Lázaro, foi ensinado o conceito de qualidade de software e seus aspectos como a medição da qualidade e a User Experience (UX) - experiência do usuário em português. Morais (2010), descreve qualidade de software “como um conjunto de características a serem satisfeitas, de modo que o produto de software atenda às necessidades de seus usuários”.

Devido ao processo de globalização e consequentemente aumento de empresas concorrentes, a produção de software de boa qualidade tornou-se um pré-requisito para que as empresas se mantenham competitivas no mercado garantindo sua sobrevivência. Nem sempre os softwares desenvolvidos alcançam um bom nível de satisfação, dessa forma, é preciso considerar a qualidade em todo o processo de desenvolvimento (Morais, 2010).

A experiência do usuário, segundo Amaral (2024, p. 17), “engloba todos os aspectos da interação do usuário com um produto ou serviço. Isso inclui a facilidade de uso, a eficiência, a satisfação emocional e a percepção geral do produto”.

Para alcançar uma excelente experiência do usuário é preciso considerar vários fatores: compreender quais as necessidades do usuário em relação ao sistema; planejar um design centrado no usuário desde o início do projeto; incluir *feedback* claro e imediato para as ações do usuário; rapidez e eficiência do sistema; garantir acessibilidade e que o sistema forneça boa usabilidade, isto é, sistemas que não sejam difíceis de utilizar, que sejam intuitivos (Amaral, 2024).

O projeto do 2º semestre consistiu em desenvolver, como trabalho avaliativo, um formulário web de inscrição para um evento de conferência de tecnologia. O objetivo consistiu em desenvolver uma interface intuitiva e responsiva com campos de formulário e botões de ação bem definidos. Como funcionalidade implementou-se a validação dos campos de formulário. Incluindo *feedbacks* ao usuário e aplicação de máscara para entrada de dados para campos como e-mail e telefone. Considerando os aspectos da qualidade de software desde o início do desenvolvimento. Sendo, portanto, uma forma de aplicar os conhecimentos teóricos da disciplina de IHC. O

projeto desenvolvido com HTML, CSS e JavaScript foi baseado nos recursos do *framework* Bootstrap que proporcionou rapidez para a criação do *layout* responsivo, além de facilitar a validação e simulação da coleta de informações. O Visual Studio Code foi o ambiente de desenvolvimento escolhido.

2.2 TECNOLOGIAS UTILIZADAS NO PROJETO

2.2.1 HTML, CSS e JavaScript

HTML significa linguagem para marcação de hipertexto. Entende-se por Hipertexto todo texto inserido em um documento para a *web* que possibilite a interligação com outros documentos da *web* (*links*). Nos dias de hoje, o HTML é uma linguagem para marcação de conteúdos *web* em geral. Não apenas hipertextos, mas também imagens, vídeos, gráficos, sons e conteúdo não textuais (hipermídia) (Silva, 2018). O CSS significa folhas de estilo em cascata. Enquanto o HTML é responsável pela marcação e estruturação de conteúdo, o CSS cuida da apresentação dos elementos da página. Ou seja, todo o aspecto visual de um documento: cores de fontes, tamanhos de texto, posicionamentos e outros (Silva, 2019). De acordo com Guedes et al. (2021, p. 2):

> A linguagem JavaScript é uma das bases do desenvolvimento de sites e sistemas *web* em conjunto com o HTML e o CSS. O JavaScript é o responsável por deixar o conteúdo da aplicação mais dinâmico, melhorando a interação com o usuário, criando animações, validando funções, dentre outros recursos que somente a linguagem é capaz de fazer em um sistema.

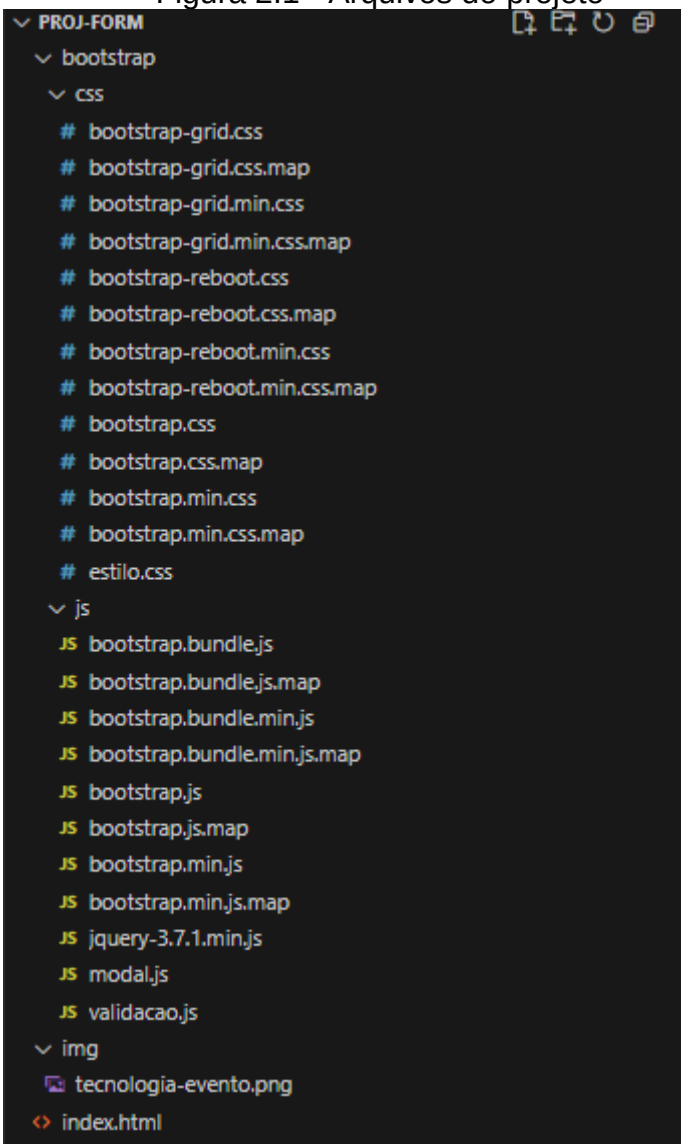
2.2.2 Framework Bootstrap

“Bootstrap é um dos mais populares *frameworks* para construção de sites responsíveis compatíveis tanto com computadores *desktop*, *smartphones* ou dispositivos com telas de diversos tamanhos” (Mariano, 2022a, p. 8).

De acordo com o site oficial em português, Bootstrap (2025), para a utilização do Bootstrap pode-se inserir na estrutura HTML básica de um site os *links* do BootstrapCDN ou então baixar os arquivos CSS e JavaScript compilados e executá-

los localmente. A Figura 2.1 mostra todos os arquivos utilizados para o desenvolvimento do formulário de inscrição.

Figura 2.1 - Arquivos do projeto

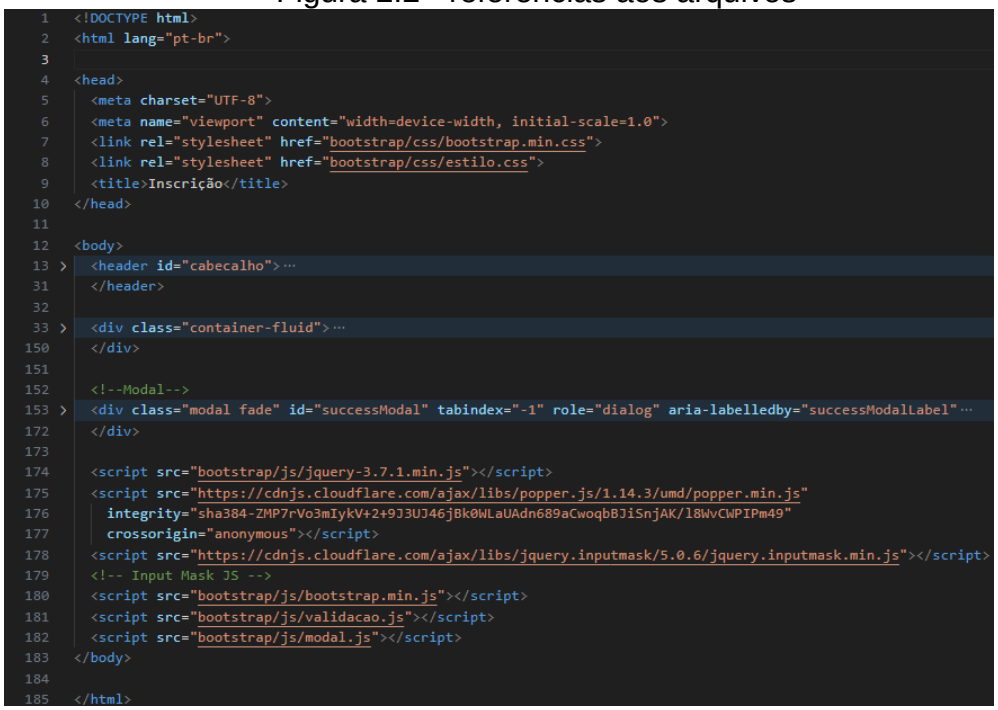


Fonte: Elaborada pelo autor, 2025

Na pasta do CSS ficam os arquivos dos estilos visuais dos componentes do Bootstrap. Deve-se importar o arquivo bootstrap.css ou o arquivo bootstrap.min.css. Na pasta do JavaScript estão os arquivos com as funções dos componentes como, por exemplo, os Menus. Basta importar o arquivo bootstrap.js ou o arquivo bootstrap.min.js para usar tudo o que o *framework* oferece. Cada pasta possui a versão padrão dos arquivos (*.css e *.js), e uma versão com os arquivos minificados, isto é, uma versão “compactada” (*.min.css e *.min.js) nas quais espaços, comentários e caracteres desnecessários são removidos para reduzir o tamanho. Recomenda-se

utilizar a versão “minificada” quando for colocar o projeto em produção (Monteiro et al., 2020). As importações dos arquivos são por meio das *tags* `<link>` e *tags* `<script>`. A Figura 2.2 é um print do projeto com todas as importações necessárias.

Figura 2.2 - referências aos arquivos



```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">
  <link rel="stylesheet" href="bootstrap/css/estilo.css">
  <title>Inscrição</title>
</head>

<body>
  <header id="cabecalho">···
  </header>

  <div class="container-fluid">···
  </div>

  <!--Modal-->
  <div class="modal fade" id="successModal" tabindex="-1" role="dialog" aria-labelledby="successModalLabel"···
  </div>

  <script src="bootstrap/js/jquery-3.7.1.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
    integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
    crossorigin="anonymous"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.inputmask/5.0.6/jquery.inputmask.min.js"></script>
  <!-- Input Mask JS -->
  <script src="bootstrap/js/bootstrap.min.js"></script>
  <script src="bootstrap/js/validacao.js"></script>
  <script src="bootstrap/js/modal.js"></script>
</body>

</html>
```

Fonte: Elaborada pelo autor, 2025

2.3 CONCEITOS IMPORTANTES

2.3.1 Responsividade

> *Design* Responsivo é um princípio de desenvolvimento para *Web* cujo objetivo é adaptar o *layout* das páginas a qualquer dispositivo, tela e resolução, com objetivo de garantir a boa experiência do usuário, possibilitando navegação e leitura confortáveis sem comprometer o conteúdo (Silva, 2014, p. 2).

O *design* responsivo não se trata apenas da adaptação do *layout* ao tamanho da tela. Um *layout* responsivo deve ser capaz de responder as características do dispositivo ao qual é servido. No sentido de movimentar-se expandindo e contraindo o *layout* com a finalidade de adaptar-se de forma acessível e usável nos mais variados tamanhos de tela (Silva, 2014).

2.3.2 Grid layout no Bootstrap

De acordo com Mariano (2022a), o sistema de grade (*grid system*) do Bootstrap divide a página em 12 colunas, permitindo posicionar elementos com precisão. Para estruturar uma página com *grids* é simples: utiliza-se as classes Contêiner (`.container`), Linha (`.row`) e Coluna (`.col-*`).

> Note que, neste caso, o * indica quantos espaços de colunas queremos alocar para determinado elemento (lembrando que o máximo é 12). Por exemplo, se quisermos dividir uma página em duas colunas, podemos alocar um tamanho 6 para cada coluna" (Mariano, 2022a, p.15).

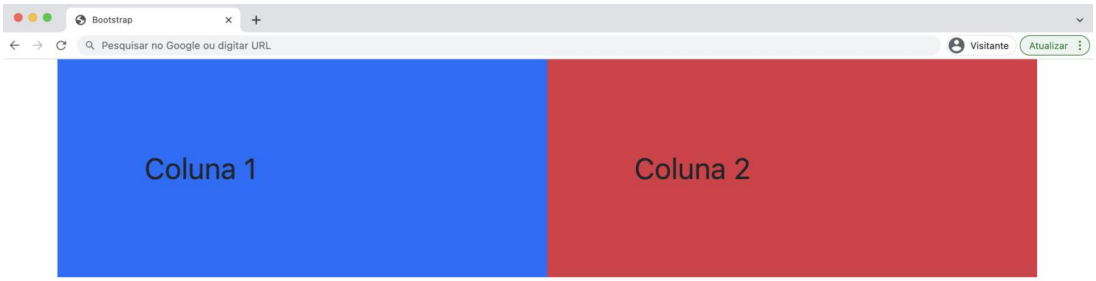
Figura 2.3 - Exemplo de código para implementar duas colunas

```
1 <div class="container">
2 <div class="row">
3 <div class="col-6 bg-primary p-5">Coluna 1</div>
4 <div class="col-6 bg-danger p-5">Coluna 2</div>
5 </div>
6 </div>
```

Fonte: Mariano, 2022b

Note que a `div` de classe `.container` delimita todo o conteúdo: uma `div` de classe `.row` servindo como linha e duas divs de classe `.col-6` (cada uma ocupa 50% do contêiner) dentro dessa mesma linha. A soma dos valores das colunas dentro de uma linha deve ser igual a 12. Valores maiores que esse provocam uma quebra para uma outra linha. As classes `bg-primary` e `bg-danger` definem a cor de fundo de cada coluna. Já a classe `p-5` aplica um espaçamento interno (Mariano, 2022a). Observe o resultado do código da Figura 2.3 na Figura 2.4:

Figura 2.4 - Página *web* com duas colunas



Fonte: Mariano, 2022b

O Bootstrap divide cada elemento em 12 colunas. Logo, seria possível dividir a coluna 1, por exemplo, em 12 outras partes. Pode-se definir pontos de quebra para as colunas. Supondo que em uma tela grande (`xl`) deseja-se manter duas colunas ocupando toda a tela, e, nas telas pequenas (`sm`) cada coluna deve ocupar 100% do espaço disponível. Cria-se esses efeitos com a classe: `.col-[ponto-de-quebra]-[valor]`. No exemplo citado, as classes das colunas ficariam: `col-xl-6 col-sm-12` (Mariano, 2022a). Pode-se observar na Figura 2.5 os sufixos de tamanho do Bootstrap.

Figura 2.5 - Sufixos de tamanho do Bootstrap

xs	Extra small <576px	portrait mobile
sm	Small ≥576px	landscape mobile
md	Medium ≥768px	portrait tablets *navbar collapse*
lg	Large ≥992px	landscape tablets
xl	Extra large ≥1200px	laptops, desktops, TVs

Fonte: Lett, 2024

2.4 DESENVOLVIMENTO DO FORMULÁRIO DE INSCRIÇÃO

2.4.1 Layout da página

A página *web* do formulário de inscrição é composta por duas principais partes: o cabeçalho e duas colunas. Uma coluna é composta por um título, um subtítulo, uma imagem e um parágrafo curto. Já a segunda coluna contém os campos de formulário e o botão de enviar. Na Figura 2.6 será apresentado o código HTML para implementar essa estrutura:

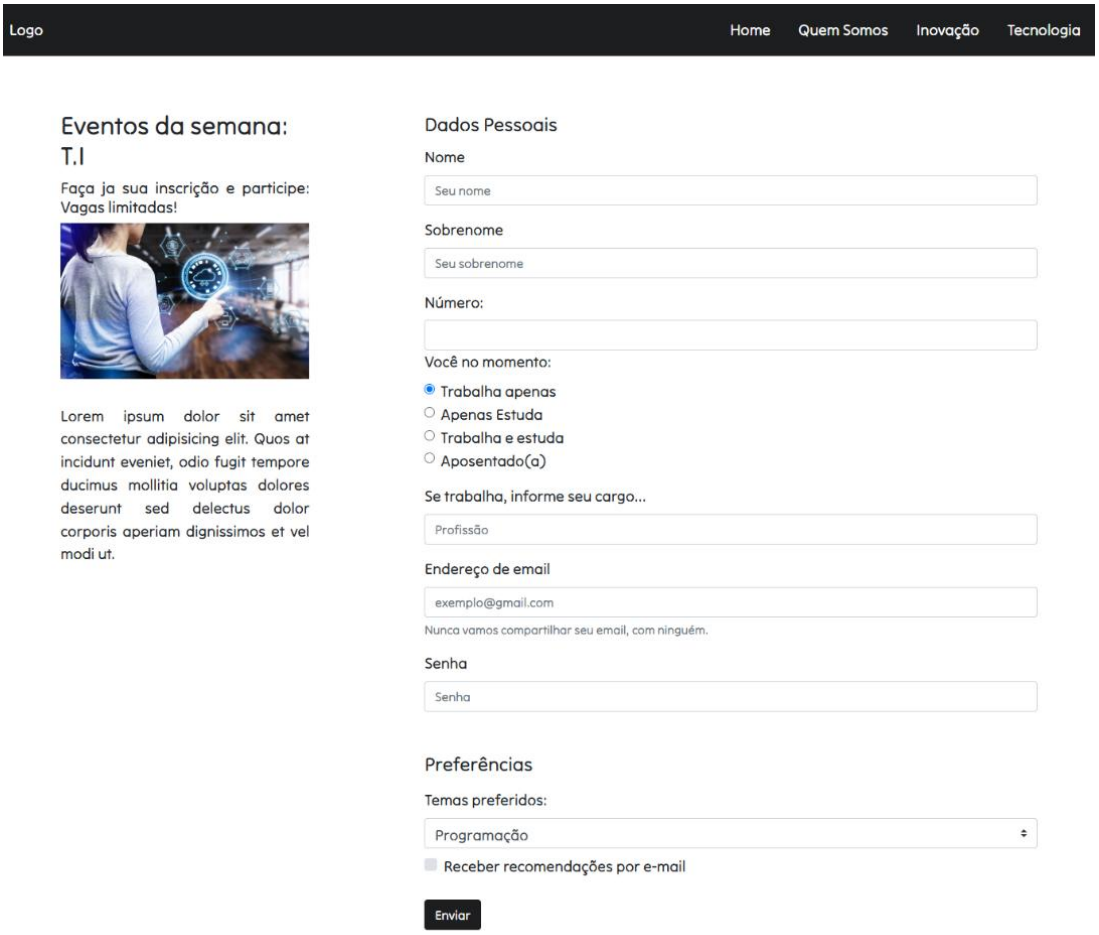
Figura 2.6 - Estrutura da página

```
<body>
  <header id="cabecalho">···
  </header>

  <div class="container-fluid">
    <div class="row">
      <!--Coluna do título, subtítulo, imagem e parágrafo-->
      <div class="col-md-3 col-sm-12 col-lg-3 col-xl-3">···
      </div>

      <!--Coluna do Formulário-->
      <div class="col-md-7 col-sm-12 col-lg-7 col-xl-7">···
      </div>
    </div>
  </div>
```

Fonte: Elaborada pelo autor, 2025

No projeto, a `div` de classe `.container-fluid` delimita todo o conteúdo: duas colunas (divs de classe `.col`) dentro da mesma linha, ou seja, dentro da `div` de classe `.row`. A depender da resolução de tela, as colunas ocupam tamanhos e posições diferentes na tela. O cabeçalho também muda conforme o dispositivo. Pode-se observar o *design* da página em dispositivos de tamanho médio ou maiores na Figura 2.7 e o *design* da página em telas pequenas na Figura 2.8:

Figura 2.7 - *Design* do formulário *web* em telas de tamanho médio, grande e extra-grande



Fonte: Elaborada pelo autor, 2025

Figura 2.8 - *Design* do formulário *web* em telas pequenas

Fonte: Elaborada pelo autor, 2025

Nota-se que para telas de tamanho médio ou maiores (`>= 768px`), sufixos de tamanho (`md`, `lg` e `xl`), o *layout* se comporta da seguinte forma: o cabeçalho é apresentado da forma padrão: um retângulo com os nomes das seções do site disponibilizados na horizontal. Na linha do contêiner, uma div `.col` ocupa três colunas (`col-md-3`) e a outra ocupa sete colunas (`col-md-7`). Restando 2 colunas vazias (completando as 12). Para telas de dispositivos pequenos (`sm`) e extra pequenos (`xs`), o cabeçalho é encurtado disponibilizando os *links* de navegação na vertical. Quanto as divs `.col`, cada um ocupa 100% do espaço disponível (`col-sm-12`) ficando uma abaixo da outra.

2.4.2 Formulário

Implementou-se a estilização dos campos de formulário e validação dos campos por meio de recursos do Bootstrap. A estrutura HTML do formulário no Bootstrap segue o padrão apresentado na Figura 2.9:

Figura 2.9 - Código HTML do formulário com recursos do Bootstrap

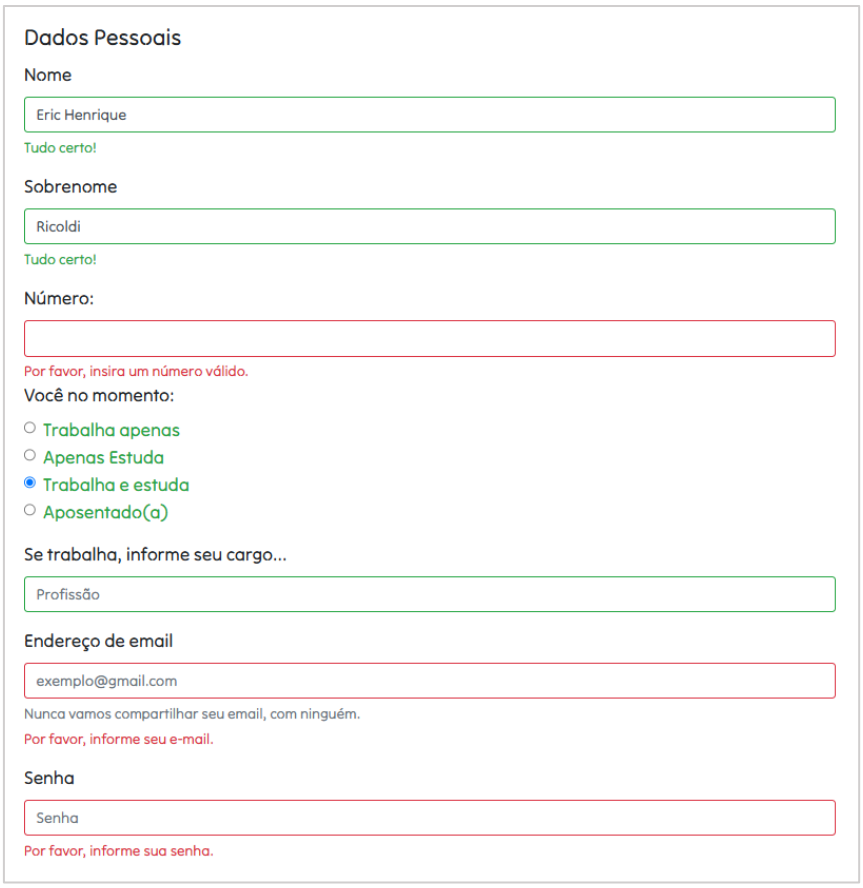
```html
<form type="submit" class="needs-validation" id="cadastroForm" novalidate>
  <fieldset>
    <legend>Dados Pessoais</legend>
    <!--Campo de Nome-->
    <div class="form-group">
      <label for="exampleInputName1">Nome</label>
      <input type="text" class="form-control" id="nome" aria-describedby="" placeholder="Seu nome" required>
      <div class="valid-feedback">
        Tudo certo!
      </div>
      <div class="invalid-feedback">
        Por favor, informe seu nome.
      </div>
    </div>

    <!--Campo de Sobrenome-->
    <div class="form-group">
      <label for="exampleInputLastName1">Sobrenome</label>
      <input type="text" class="form-control" id="sobrenome" aria-describedby="" placeholder="Seu sobrenome"
        required>
      <div class="valid-feedback">
        Tudo certo!
      </div>
      <div class="invalid-feedback">
        Por favor, informe seu sobrenome.
      </div>
    </div>
```

Fonte: Elaborada pelo autor, 2025

"Campos de formulário textuais como `<input>`, `<select>` e `<textarea>` são estilizados com a classe `.form-control`. Ela possui estilos para aparência, estado de foco e muito mais" (Bootstrap, 2025). A validação de formulário HTML usando Bootstrap é aplicada por meio de duas pseudo-classes CSS: *invalid* e *valid*. São mensagens de *feedback* customizadas que aparecem logo abaixo dos campos de entrada.

Essas classes visuais são controladas por meio de um código JavaScript no qual impede o envio do formulário se houver campos inválidos e adiciona a classe do Bootstrap `was-validated`, que faz com que o *framework* mostre automaticamente os estilos de erro (`invalid-feedback`) e sucesso (`valid-feedback`) nos campos do formulário. Pode-se escolher quando ativar as classes de *feedback* (tipicamente, depois do envio de formulário). A utilização do atributo booleano `novalidate` no `<form>` é necessário pois, desativa as dicas padrões do *browser*, mas continua tendo acesso as APIs de validação de formulário, no JavaScript (Bootstrap, 2025). O código JavaScript mencionado é o arquivo chamado `validacao.js`. Nele, também, implementou-se máscara para entrada de dados nos campos: e-mail e telefone utilizando a biblioteca *input mask* JS. Observa-se o funcionamento das validações na Figura 2.10.

Figura 2.10 - Mensagens de *feedback* em formulários do Bootstrap



Fonte: Elaborada pelo autor, 2025

2.4.3 Janela Modal

Quando todos os campos obrigatórios estão preenchidos e o botão de enviar é clicado, abre-se um *modal* de sucesso para simular a confirmação de envio das informações. O *modal* é uma pequena janela posicionada acima de todas as coisas do documento para exibir informações importantes ou solicitar ações específicas do usuário (Bootstrap, 2025). Sendo útil para alertas, confirmações e exibir detalhes adicionais sem que o usuário precise sair da página. Verifica-se, na Figura 2.11, o código HTML do *modal* de confirmação:

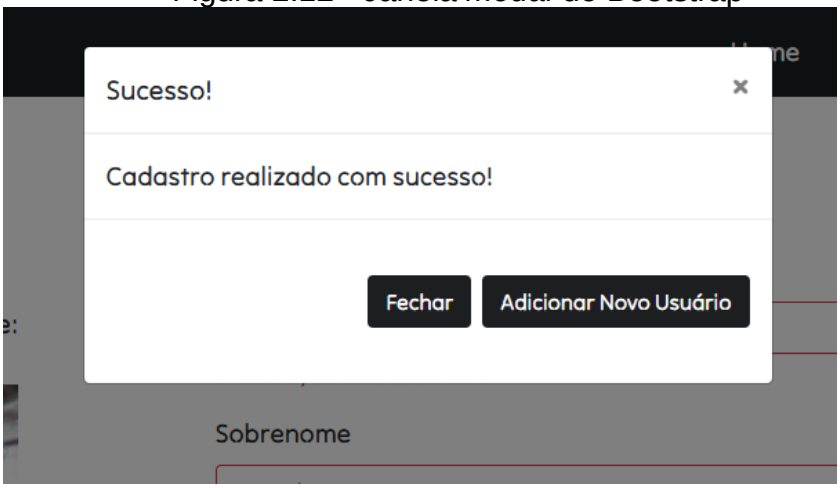
Figura 2.11 - Código HTML da janela *modal*

```
<!--Modal-->
<div class="modal fade" id="successModal" tabindex="-1" role="dialog"
aria-labelledby="successModalLabel"
  aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="successModalLabel">Sucesso!</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Fechar">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        Cadastro realizado com sucesso!
      </div>
      <div class="modal-footer">
        <button type="button" class="btn" data-dismiss="modal">Fechar</button>
        <button type="button" class="btn" id="addNewUserBtn">Adicionar Novo Usuário</
        button>
      </div>
    </div>
  </div>
</div>
```

Fonte: Elaborada pelo autor, 2025

É um componente pronto disponibilizado pelo Bootstrap. É dividido em três partes: *header* (cabeçalho), *body* (o corpo da janela *modal*) e o *footer* (rodapé). Observa-se a janela *modal* na Figura 2.12:

Figura 2.12 - Janela *modal* do Bootstrap



Fonte: Elaborada pelo autor, 2025

O *modal* é acionado por meio do arquivo JavaScript `modal.js`. Ele é responsável, também, por validar os campos de formulário. Na ocorrência de campos obrigatórios vazios exibe um alerta de erro. Caso contrário exibe o modal de sucesso (`#successModal`) na tela e limpa os campos de entrada. A Figura 2.13 mostra o código JavaScript.

Figura 2.13 - Código JavaScript para acionar o *Modal*

Fonte: Elaborada pelo autor, 2025

2.5 METODOLOGIAS APLICADAS NO PROJETO E PESQUISA DE SATISFAÇÃO

2.5.1 Contextualização e definições

De acordo com Morais (2010) “a experiência do usuário, além das qualidades técnicas do *software* é um fator determinante para a construção de sistemas de maior qualidade”. Para o desenvolvimento do formulário *web* de inscrição aplicou-se os conhecimentos discutidos nas aulas de IHC: usabilidade, funcionalidade, *design* centrado no usuário (DCU) e psicologia e teoria das cores.

A usabilidade é um aspecto da qualidade de software que envolve o funcionamento correto do software, segurança, facilidade de utilização e facilidade de integração com outros sistemas. Já a funcionalidade é o conjunto de funções do sistema que satisfazem as necessidades explícitas e implícitas do usuário (Morais, 2010). Consoante a Azevedo e Gibertoni (2020, p. 296):

O DCU “consiste em um conjunto de técnicas, métodos, procedimentos e processos, sendo uma filosofia que coloca o usuário no centro do processo de desenvolvimento de uma forma intensa, conveniente e sistemática”. De acordo com

Salah, Paige e Cairns (2014) “a usabilidade de um produto é resultante do trabalho pautado no DCU”. Quanto a psicologia e teoria das cores:

“No design, as cores desempenham um papel fundamental na identidade visual, acessibilidade e experiência do usuário. Cada tom pode evocar emoções específicas, influenciar percepções e até direcionar o comportamento dos usuários” (Teoria, 2025).

2.5.2 Métodos

Para produzir uma interface com boa usabilidade, com foco na pessoa usuária e que cumprisse sua principal função (simular o funcionamento de um formulário *web* de inscrição) aplicou-se os seguintes métodos: criação de um *layout* limpo com espaços em branco posicionados estrategicamente e foco em destacar os elementos essenciais (campos de formulário e botões de ação); aplicou-se um *layout* fluido e responsivo por meio do *framework* Bootstrap, permitindo, assim, acessar o formulário em diferentes tamanhos de tela; a aplicação de máscara para campos como email, telefone e senha junto das mensagens de *feedback* (alerta de campo(s) vazio(s) e janela *modal*) contribuem para uma boa e completa experiência de usuário. Branco e preto são as cores predominantes do *design*. O branco é presente em *designs* modernos e *clean*, enquanto o preto transmite sofisticação e elegância (Teoria, 2025).

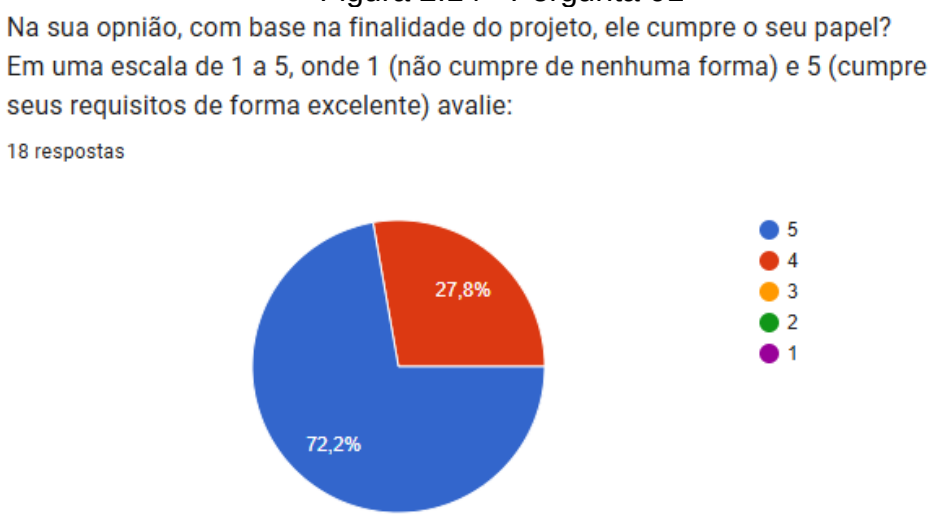
2.5.3 Pesquisa de satisfação

Como forma de validar o projeto desenvolvido com base no conteúdo sobre qualidade de software nas aulas de IHC, elaborou-se um questionário online (Google Forms) onde 18 alunos do curso de ADS da Fatec Lins responderam perguntas sobre o projeto. Os estudantes tiveram acesso à página *web* do projeto e puderam interagir com a interface. O objetivo foi averiguar se o que foi desenvolvido cumpriu ou não seu objetivo e medir o nível de satisfação do usuário em relação a usabilidade, funcionalidade e a interface.

As perguntas do questionário foram de múltipla escolha. A primeira pergunta foi para saber se o projeto cumpria ou não seu papel: simular a coleta e envio de informações para inscrição de pessoas. Definiu-se uma escala de 1 a 5, na qual 1

(significa que não cumpre de nenhuma forma) e 5 (cumpre seus requisitos de forma excelente). Observa-se, na Figura 2.14, o *feedback* recebido.

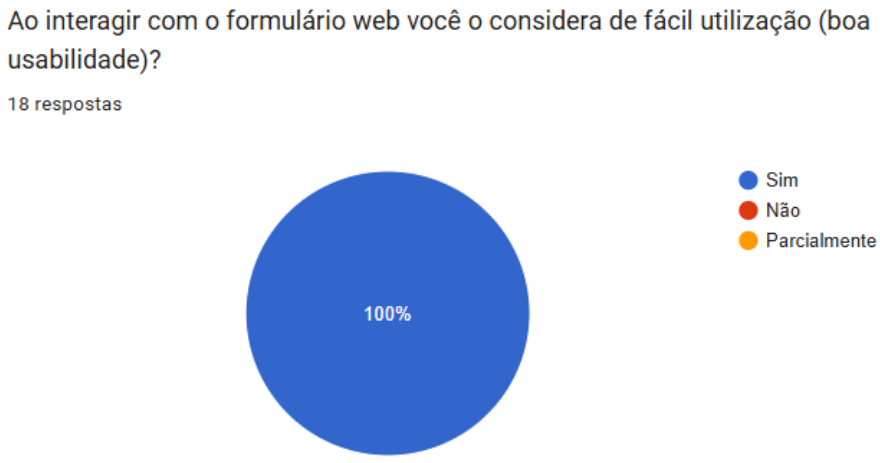
Figura 2.14 - Pergunta 01



Fonte: Elaborada pelo autor, 2026

A segunda pergunta foi em relação a usabilidade. Verifica-se as respostas na Figura 2.15:

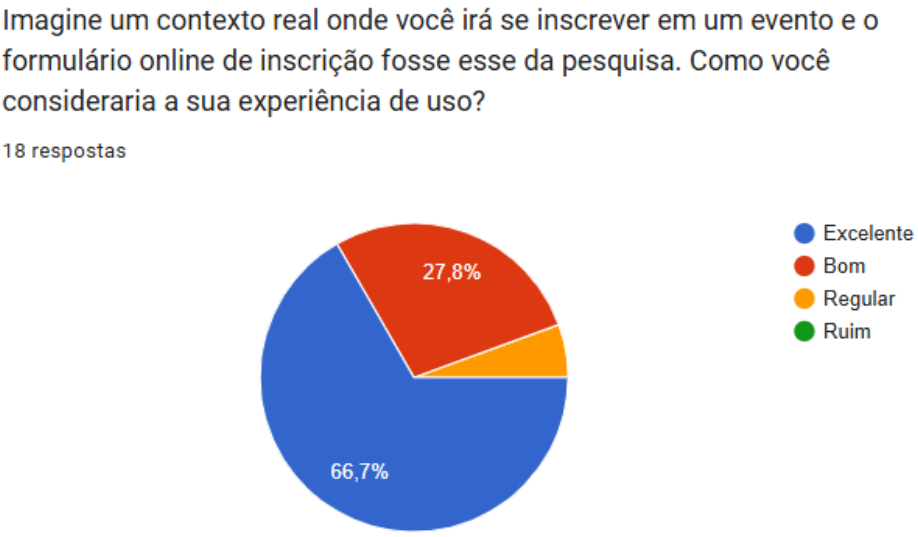
Figura 2.15 - Pergunta 02



Fonte: Elaborada pelo autor, 2026

Supondo que o projeto do formulário web estivesse realmente sendo utilizado na internet, como seria a experiência de uso dos usuários? A terceira pergunta tem o objetivo de responder a essa dúvida. Observa-se na Figura 2.16 o *feedback* recebido:

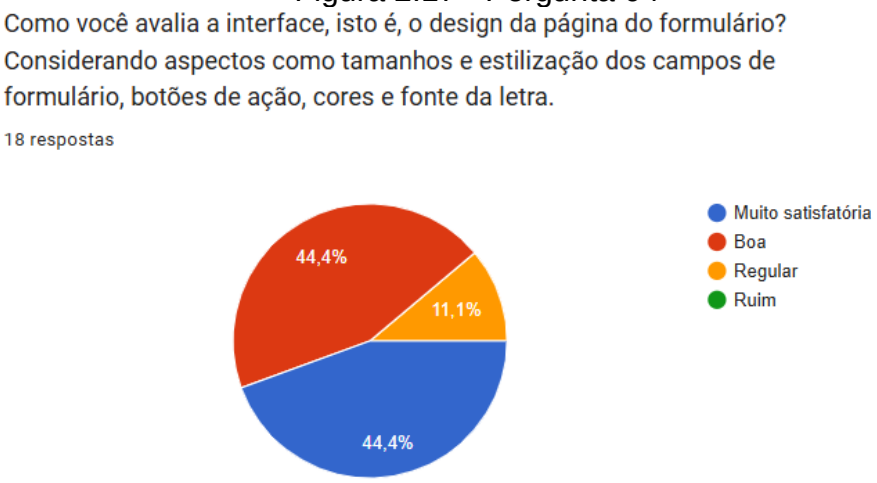
Figura 2.16 - Pergunta 03



Fonte: Elaborada pelo autor, 2026

A quarta pergunta foi para avaliar o nível de satisfação dos usuários em relação a interface. Observa-se os resultados colhidos na Figura 2.17:

Figura 2.17 - Pergunta 04



Fonte: Elaborada pelo autor, 2026

Com base nas respostas colhidas do questionário pode-se concluir que o projeto atingiu um nível de satisfação bom dentro dos critérios avaliados: usabilidade, funcionalidade e *design* centrado no usuário. Dessa forma, validando os conhecimentos da disciplina de IHC aplicados durante o desenvolvimento do projeto.

2.6 CONCLUSÃO

Desenvolver sistemas de qualidade é fundamental para as organizações. E, a implementação de uma interface centrada no usuário é um importante fator para gerar boa experiência de usuário. O projeto configurou-se como uma experiência prática onde, o aluno, precisou planejar como proporcionar satisfação por meio da interface. Considerando-se a qualidade de software em todo o processo de desenvolvimento. Além de se trabalhar o senso crítico do estudante, o projeto, contribuiu para o conhecimento técnico do discente ao utilizar tecnologias do desenvolvimento *web* como o *framework* Bootstrap, a linguagem de programação JavaScript e o HTML e CSS.

3 PROJETO DO 3º SEMESTRE: SISTEMA PARA GESTÃO DE UMA LIVRARIA DESENVOLVIDO NA LINGUAGEM C

3.1 CONTEXTUALIZAÇÃO DA MATÉRIA DE ESTRUTURA DE DADOS

Na disciplina de estrutura de dados, ministrada pelo professor Júlio Fernando Lieira, aprendeu-se sobre as estruturas de dados. Por meio da construção de programas (exercícios), na linguagem C, pode-se compreender, na prática, a utilização de estruturas de dados fundamentais como: lista, pilha, *structs* (estruturas), ponteiros e arquivos. Cada uma dessas estruturas pode ser utilizada para criar diferentes tipos de soluções, algoritmos e sistemas computacionais. Para o desenvolvimento do sistema de gerenciamento de uma livraria, na linguagem C, utilizou-se as estruturas de dados arquivos e *structs*. O uso do sistema de arquivos possibilitou criar, manter, alterar, excluir e listar informações. Funcionando como um "banco de dados" do sistema.

3.1.1 O que é estrutura de dados e sua importância

Conforme Amoasei (2022) "as estruturas de dados, junto com o estudo de algoritmos, fazem parte dos fundamentos da programação". Os dados representam uma unidade ou um elemento de informação. Podem ser acessados por meio de um identificador (uma variável, por exemplo). Normalmente, utiliza-se os dados de forma conjunta (Amoasei, 2022). A forma como estes dados serão agregados e organizados depende muito de como serão utilizados e processados, levando-se em consideração, por exemplo, a eficiência para buscas, o volume dos dados trabalhados, a complexidade da implementação e a forma como os dados se relacionam. Estas diversas formas de organização são as chamadas estruturas de dados (Amoasei, 2022).

Consoante a Sorage (2023) o conhecimento sobre estruturas de dados é fundamental para a construção de *softwares* eficientes e escaláveis. Sem um conhecimento sólido sobre estrutura de dados o desenvolvedor fica sujeito a desenvolver soluções pouco escaláveis, ineficientes e com má otimização. É importante destacar que linguagens de programação são um meio para aplicarmos lógica e algoritmos para criar uma solução. Elas podem mudar e evoluir com o tempo,

já as estruturas de dados são conhecimentos consistentes e que não mudam com o tempo. “Elas são o coração da lógica e do raciocínio que impulsionam os algoritmos” (Sorage, 2023).

3.2 COMPREENDENDO AS ESTRUTURAS DE DADOS E AS TECNOLOGIAS APLICADAS NO SISTEMA DA LIVRARIA

3.2.1 A linguagem C

A linguagem C é uma linguagem de programação de alto nível. Um programa em C é baseado no uso de funções. Os programadores utilizam a coleção de funções da biblioteca padrão C, mas também, podem usufruir de funções criadas por si próprio ou criadas por outras pessoas. Algumas caraterísticas da linguagem é portabilidade, modularidade, recursos de baixo nível e simplicidade (Lacerda, 2002).

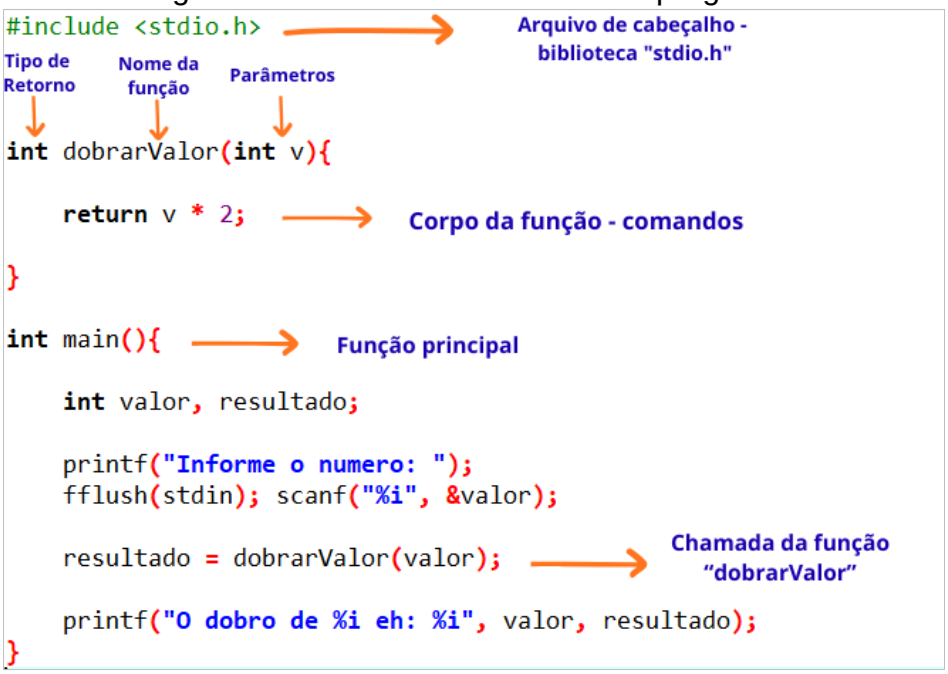
> Um programa em C é estruturado sobre funções. O programa possui uma ou várias funções, sendo que a principal, que dá inicio ao programa e chama todas as outras, é sempre chamada *main.* Além dessas, existem outras pré-programadas, que são incluídas nos programas como arquivos de cabeçalho, ou arquivos de inclusão. Todas as funções começam com { e terminam com } (Lacerda, 2002, p. 17).

3.2.2 O que são funções

Ao desenvolver um programa o desenvolvedor pode-se deparar com algum bloco de código que pode ser utilizado em outras partes do programa. No entanto, copiar o bloco para os locais necessários não é uma boa prática, pois, gera problemas como: precisar atualizar todas as cópias na ocorrência de alterações em uma delas, torna o programa mais complexo de entender e manter ao longo do tempo devido as várias cópias do mesmo código. A utilização de funções soluciona ou minimiza esses problemas. Função trata-se de um bloco de código com uma tarefa específica. Então, invés de inserir várias cópias de um bloco de código no programa, basta colocar o bloco dentro de uma função. Dessa forma, quando for necessário, basta chamar a função para executar o código automaticamente (Balieiro, 2015).

Observa-se a estrutura básica de um programa em C que exemplifica a criação de uma função na Figura 3.1:

Figura 3.1 - Estrutura básica de um programa em C



Fonte: Elaborada pelo autor, 2025

3.2.3 Ponteiros

Ponteiro é uma variável que contém o endereço de outra variável. A forma de declaração de um ponteiro é: `tipo_do_ponteiro *nome_da_variavel;`

O asterisco (*) indica ao compilador que aquela variável não vai guardar um valor, mas sim um endereço para aquele tipo especificado. Ao utilizar um ponteiro é importante apontá-lo para algum lugar conhecido. Pode-se atribuir o endereço de memória de uma variável a um ponteiro por meio do operador &. No exemplo:

```
int count;
int *pt;
pt = &count;
```

Nota-se que há uma variável do tipo inteiro chamada *count*. A declaração, `int *pt`, cria um ponteiro para uma variável do tipo inteiro. Já a expressão: `pt = &count`, atribui ao ponteiro `pt` o endereço de memória da variável *count*. É possível modificar o conteúdo da variável *count* por meio do ponteiro com o uso do operador (*). A expressão `*pt` é equivalente ao próprio *count* (Lacerda, 2002).

3.2.4 Estruturas (*structs*)

Schildt (1996, p. 167) descreve o tipo de dado estrutura, na linguagem C, como “uma coleção de variáveis referenciadas por um nome, fornecendo uma maneira conveniente de se ter informações relacionadas agrupadas”. As variáveis que compõe uma estrutura são chamadas membros da estrutura. Também conhecidas como elementos ou campos. Geralmente, esses campos possuem uma relação lógica. Uma estrutura que representa uma lista postal, por exemplo, faz sentido conter campos como nome e endereço. A palavra *struct* define que será criado um modelo de estrutura. O identificador da estrutura (nome) é também seu especificador de tipo. Para declarar uma variável do tipo *struct*, escreve-se:

`struct nome_estrutura nome_variavel` (Schildt, 1996). Verifica-se um exemplo de *struct* em um programa em C na Figura 3.2:

Figura 3.2 - Programa: *struct* TipoPessoa

```c
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

struct TipoPessoa {
    char nome[50];
    int idade;
    char sexo;
};

int main(){
    struct TipoPessoa pessoa1;

    printf("informe o nome: ");
    fflush(stdin); gets(pessoa1.nome);

    printf("\ninforme a idade: ");
    fflush(stdin); scanf("%i", &pessoa1.idade);

    printf("\ninforme o sexo (M/F) ou (m/f): ");
    fflush(stdin); scanf("%c", &pessoa1.sexo);

    printf("\n\n------Informacoes da pessoa------");
    printf("\nNome: %s", pessoa1.nome);
    printf("\nIdade: %i", pessoa1.idade);
    printf("\nSexo: %c", pessoa1.sexo);
}
```

Fonte: Elaborada pelo autor, 2025

O programa cria uma *struct* cujo identificador/nome é “TipoPessoa” com três campos: nome, idade e sexo. Dentro da *main* declarou-se:

`struct TipoPessoa pessoa1;`

A declaração cria uma variável chamada pessoa1 do tipo *struct* TipoPessoa. Quando declaramos uma variável de estrutura como pessoa1, por exemplo, Schildt

(1996) explica que o compilador aloca automaticamente memória suficiente para armazenar os campos da estrutura. E para referenciar um elemento específico da estrutura basta escrever o nome da variável do tipo *struct* seguido por um ponto e pelo nome do elemento:

```
nome_da_estrutura.nome_do_elemento
```

No programa, para atribuir um valor ao campo nome da variável pessoa1, por exemplo, declarou-se:

```
gets (pessoa1.nome);
```

O comando faz a leitura da *string* fornecida pelo usuário e armazena o valor no campo nome da variável pessoa1. Para imprimir o valor do campo nome ou qualquer outro campo da *struct* basta referenciar o elemento desejado dentro da função `printf()`.

3.2.5 Arquivos em C

Quando um programa em C é finalizado por algum motivo, como desligar o computador, por exemplo, as informações são perdidas. Isso ocorre porque a memória RAM é volátil. Uma forma de não perder as informações é utilizar os chamados arquivos (Gotardo, 2015). “As funções para manipulação de arquivos em alto nível são bufferizadas, ou seja, mantém uma região de memória com os dados antes de enviá-los para o destino” (Gotardo, 2015, p. 169).

Na linguagem C pode-se trabalhar com arquivos bufferizados por meio de *Streams* (fluxos de *bytes*). As *streams* podem ser textuais (sequência de caracteres) ou binárias (sequência de *bytes*). Existe funções para manipulação de arquivos texto e binário (Gotardo, 2015).

Os arquivos não podem ser manipulados diretamente, é necessário que exista uma variável para referenciá-lo. Esta variável deve ser do tipo `FILE*`. Sempre é necessário criar uma variável deste tipo para poder utilizar o arquivo. Todas as funções de manipulação de arquivo necessitam de uma variável deste tipo para poder manipular o arquivo (Gotardo, 2015).

A declaração de uma variável que manipula arquivos, segundo Gotardo (2015), é da seguinte forma: `FILE* arquivo;`

Com a declaração da variável que referencia o arquivo, pode-se usar a função *`fopen()`* para abrí-lo. Sua sintaxe é:

```
nome_variavel = fopen("nome_arquivo.extensão"."modo_abertura");
```

- **nome_variável**: variável do tipo `FILE` que representa o arquivo dentro do programa;
- **"nome_arquivo.extensão"**: arquivo que existe ou será criado;
- **"modo_abertura"**: como abrir, criar ou sobrescrever o arquivo.

Na operação de abertura de arquivo, "é associada uma *stream* com um arquivo específico. Quando o arquivo é aberto, informações podem ser trocadas entre a *stream* e o programa" (Gotardo, 2015).

3.2.6 Funções para manipulação de arquivos

As seguintes funções, descritas por Gotardo (2015), foram utilizadas no sistema da livraria:

- `fopen()` – abre um arquivo;
- `fclose()` – fecha o arquivo garantindo a transferência de *buffer*;
- `fread()` – lê um bloco de dados do arquivo;
- `fwrite()` – escreve um bloco de dados no arquivo;
- `fseek()` – reposiciona o ponteiro.

A função `fwrite()` é utilizada para gravar estruturas complexas em um arquivo binário, tendo quatro parâmetros:

```
fwrite(const void *elemento, size_t tamanho_elemento, size_t
n_de_elementos, FILE *arq_binario);
```

- **elemento:** ponteiro do elemento que vai ser escrito no arquivo.
- **tamanho_elemento:** tamanho em *byte* de cada elemento que vai ser escrito.
- **n_de_elementos:** número de elementos que serão escritos.
- **arq_binario:** ponteiro para o arquivo de gravação.

Já a função `fread()`, por sua vez, é usada para recuperar os dados de arquivos binários. Sua sintaxe:

```
fread(const void *elemento, size_t tamanho_elemento, size_t n_de_elementos, FILE *arq_binario);
```

Perceba que os parâmetros são os mesmos de `fwrite()`:

- **elemento:** ponteiro para um bloco do elemento onde vai ser escrito a informação buscada no arquivo;
- **tamanho_elemento:** tamanho em *byte* de cada elemento que vai ser escrito.
- **n_de_elementos:** número de elementos que serão escritos.
- **arq_binario:** ponteiro para o arquivo de leitura.

(Gotardo, 2015).

3.3 SISTEMA DA LIVRARIA

3.3.1 Arquivos de cabeçalho e nome dos arquivos

Cada função da biblioteca C padrão tem um arquivo de cabeçalho associado a ela. Deve-se incluí-los no programa usando: `#include`. O arquivo de cabeçalho `STDIO.H`, por exemplo, permite operações com arquivos (Schildt, 1996). Na Figura 3.3, nota-se, os arquivos de cabeçalho e os nomes dos arquivos binários utilizados no sistema:

Figura 3.3 - Arquivos de cabeçalho e nome dos arquivos binários

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4  #include <stdlib.h>
5
6  #define LIVROS "Livros.dat"
7  #define CLIENTES "Clientes.dat"
8  #define VENDAS "Vendas.dat"
9  #define AUTORES "Autores.dat"
10 #define RELACAO "Relacao_autor_livro.dat"
```

Fonte: Elaborada pelo autor, 2025

Percebe-se a definição de constantes invés de utilizar o nome original dos arquivos binários.

3.3.2 Estruturas (*structs*) do sistema da livraria

O sistema da livraria é composto por livros, autores e clientes. Para cada uma dessas entidades há uma estrutura *(struct)* e um arquivo binário dedicado. Observa-se, na Figura 3.4, as estruturas utilizadas no projeto:

Figura 3.4 - *Structs* do sistema da livraria

```
struct reg_livro{
  int codigo;
  char titulo[30];
  float preco;
};

struct reg_cliente{
  int codigo;
  char nome[40];
  char fone[15];
  char email[30];
};

struct reg_venda{
  int codigo; //Código da Venda
  int codcli; //Código do Cliente
  int codliv; //Código do Livro a ser vendido
  int qtde; //Quantidade vendida
  float desconto; //Desconto
};

struct reg_autor {
    int codigo; //código do autor
    char nome[30];
};

struct reg_RelAutorLivro{
    int codrelacao;
    int codautor; //código do Autor
    int codlivro; //código do Livro
};
```

Fonte: Elaborada pelo autor, 2025

A *struct reg_livro* é utilizada para gravar dados de livros no arquivo binário chamado “LIVROS”. A *struct reg_cliente* armazena dados de clientes no arquivo “CLIENTES” e *struct reg_autor* registra os dados de autores no arquivo “AUTORES”. O sistema permite associar autores a livros. Um autor pode estar associado a muitos livros e um livro pode estar associado a muitos autores. Isso cria uma relação de muitos para muitos. Sendo assim, no sistema, temos a *struct reg_RelAutorLivro* que serve para armazenar as relações entre livros e autores no arquivo “RELACAO”.

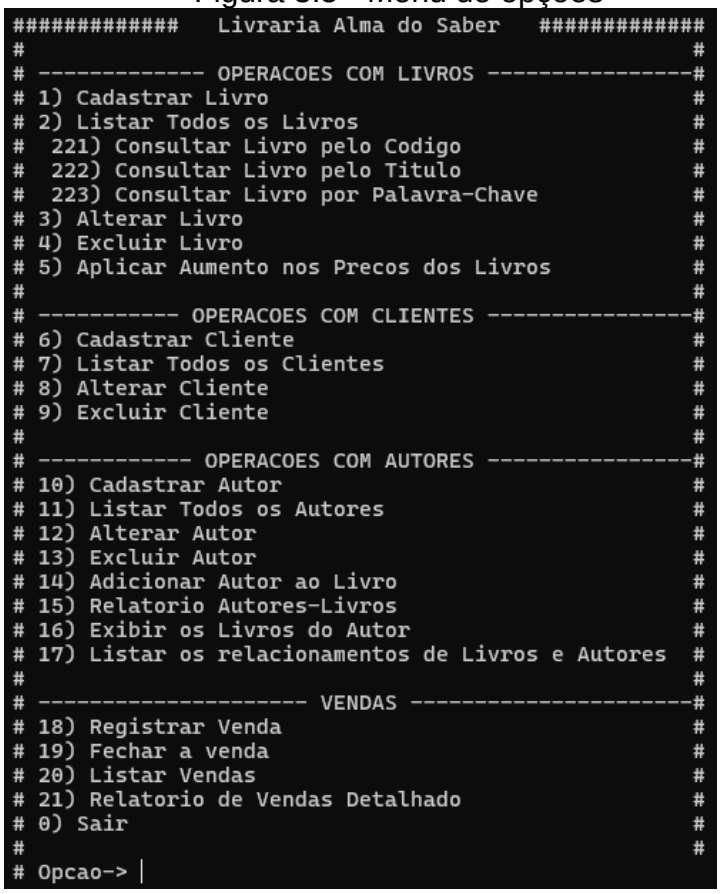
Esse arquivo, portanto, funciona como uma tabela intermediária, semelhante ao que acontece em bancos de dados relacionais, pois, armazena os códigos de autor e livro como se fossem chaves estrangeiras em uma tabela (arquivo “RELACAO”). Outra funcionalidade do sistema é registrar vendas de livros. Para tal, temos a *struct*

reg_venda que armazena as vendas no arquivo “VENDAS”. Esse arquivo também funciona como uma espécie de tabela intermediária para solucionar uma relação de muitos para muitos entre livros e clientes; nele são armazenados o código da venda (int código), as “chaves estrangeiras” código do cliente (int codcli) e código do livro (int codliv) e outros dois valores: quantidade de livros vendidos (int qtde) e desconto (*float* desconto).

3.3.3 Menu de opções

Como mencionado anteriormente, um programa em C é baseado em funções. A função principal (*main*) é responsável por iniciar o programa e chamar todas as outras funções. No sistema, a *main*, possui um menu de opções que representa as funcionalidades do sistema. Verifica-se, na Figura 3.5, o menu do sistema:

Figura 3.5 - Menu de opções



Fonte: Elaborada pelo autor, 2025

Nota-se, por exemplo, que a opção de número 1 é para cadastrar um livro e a opção 2 lista todos os livros cadastrados. Quando o usuário tecla a opção desejada e pressiona *enter* a função correspondente à opção é chamada.

3.4 FUNÇÕES DO SISTEMA DA LIVRARIA

Neste tópico descreve-se as principais funções do sistema. Que são as funções para *Create, Read, Update e Delete* (CRUD) (Miranda, 2024). As funções para operações de CRUD de livros, autores e clientes são bem similares. A lógica é a mesma, mudando apenas as *structs* e os arquivos binários utilizados. Há, também, as funções que relacionam mais de um arquivo e que também realizam operações de escrita e leitura. São elas: funções para venda e funções que relacionam livros e autores.

3.4.1 Função cadastrarLivro()

Essa função é responsável por armazenar novos registros de livros no arquivo binário chamado “LIVROS”. Observa-se seu código na Figura 3.6:

Figura 3.6 - Função para cadastro de livro

```c
void cadastrarLivro(){
  FILE *fplivro;
  struct reg_livro livro;
  char opc;

  //Entrada de Dados - Pedir os dados do Livro
  printf("\nDigite o Codigo:");
  fflush(stdin); scanf("%i",&livro.codigo);
  printf("Titulo: ");
  fflush(stdin); gets(livro.titulo);
  printf("Preco: ");
  fflush(stdin); scanf("%f", &livro.preco);

  printf("\nGravar Livro?(S/N)");
  fflush(stdin); scanf("%c",&opc);

  if ((opc!='s') && (opc!='S')){
    printf("\nOperacao Cancelada!");
    return; //Retorna ao menu principal
  }

  //Abrir o Arquivo
  fplivro = fopen(LIVROS,"ab");

  //Gravar o Registro
  fwrite(&livro,sizeof(livro),1,fplivro);

  //Fechar o Arquivo
  fclose(fplivro);

  printf("\nLivro gravado com sucesso.");
}//Fim cadastrarLivro()
```

Fonte: Elaborada pelo autor, 2025

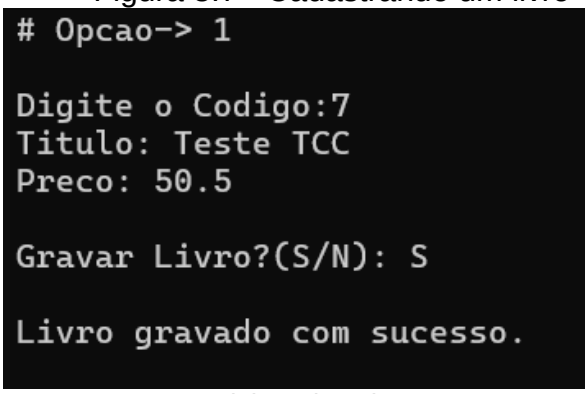
Antes de entender cada parte do código é importante saber que as informações do sistema da livraria são gravadas e lidas em arquivos binários. Essas informações são gravadas e lidas em bloco de dados, ou seja, por meio de *structs.* As declarações das variáveis nas primeiras linhas significam:

```
FILE *fplivro; // Variável para manipular o arquivo

struct reg_livro livro; // declara uma variável de estrutura

char opc; // variável de controle
```

Quando a função `cadastrarLivro()` é chamada, a primeira instrução que ela realiza é solicitar os dados do novo livro (código, título e preço) e armazena esses dados na variável de estrutura livro. Após colher as informações, o programa pede uma confirmação para gravar o livro. Se confirmado, é realizado uma operação de abertura de arquivo com `fopen()` e os registros são gravados no arquivo “LIVROS”

por meio da função `fwrite()`. Por fim, é necessário fechar o arquivo com `fclose()`. Verifica-se, na Figura 3.7, um exemplo de operação de cadastro de Livro:

Figura 3.7 - Cadastrando um livro



Fonte: Elaborada pelo autor, 2025

3.4.2 Função listarLivros()

A função listarLivros() exibe as informações de todos os livros cadastrados no arquivo binário “LIVROS”. Observa-se seu código na Figura 3.8:

Figura 3.8 – Função para listar todos os livros

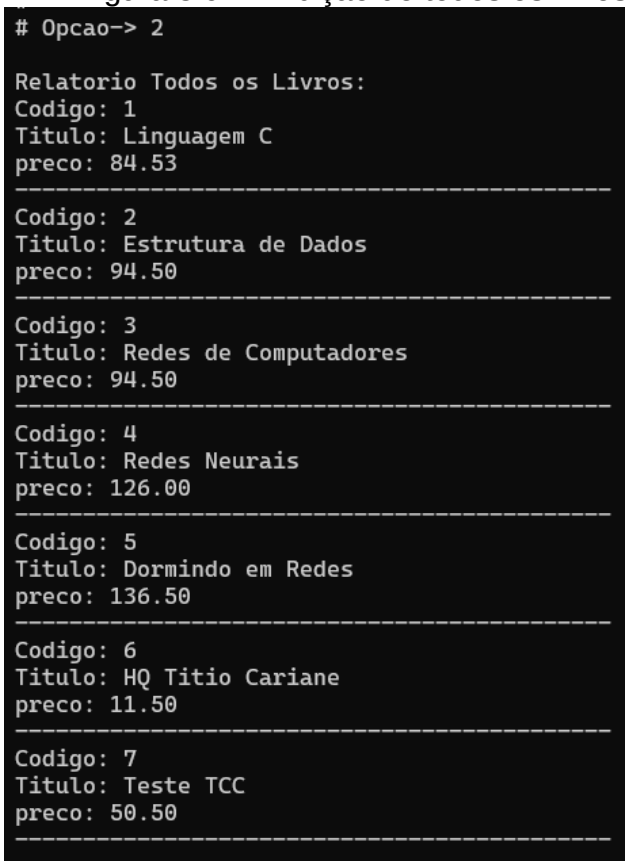
```
void listarLivros(){
  FILE *fplivro;
  struct reg_livro livro;

  //Abrir o arquivo
  fplivro = fopen(LIVROS,"rb");

  printf("\nRelatorio Todos os Livros: ");
  //Ler registro por registro e mostar na tela
  while(fread(&livro,sizeof(livro),1,fplivro)==1){
      printf("\nCodigo: %i",livro.codigo);
      printf("\nTitulo: %s",livro.titulo);
      printf("\npreco: %5.2f",livro.preco);
      printf("\n----------------------------------------");
  }
  //Fechar o Arquivo
  fclose(fplivro);

}//Fim listarLivros()
```

Fonte: Elaborada pelo autor, 2025

Percebe-se que a função `listarLivros()` também utiliza ponteiro de arquivo (`FILE *fplivro`), variável de estrutura, função `fopen()` e `fclose()`. A diferença entre a função `listarLivros()` e `cadastrarLivro()` é que no cadastro de livros o arquivo “LIVROS” é aberto para escrita, já na função de listar livros o arquivo é

aberto para recuperação dos dados. A função `fread()` dentro do laço de repetição permite exibir os dados de todos os livros cadastrados. Observa-se, na Figura 3.9, o resultado da chamada à função `listarLivros()`:

Figura 3.9 - Exibição de todos os livros



Fonte: Elaborada pelo autor, 2025

As demais funções do sistema apresentam similaridades com as funções `cadastrarLivro()` e `listarLivros()`, pois, todas utilizam variáveis *struct*, arquivos binários e as funções `fopen()`, `fclose()`, `fwrite()` e/ou `fread()`. O que varia um pouco é a lógica aplicada, as variáveis *struct* utilizadas e os arquivos binários utilizados. Sendo assim, para as próximas funções descritas neste capítulo será apresentado uma breve descrição da lógica do código de cada função e um exemplo da função sendo utilizada.

3.4.3 Função buscarLivroPeloCodigo()

Para alterar e excluir um livro é necessário saber qual o livro que o usuário do sistema deseja. Para isso, a função `buscarLivroPeloCodigo()` é utilizada por funções como `alterarLivro()`, `excluirLivro()`, funções para operações de venda e funções que relacionam livros e autores. O propósito da função `buscarLivroPeloCodigo()` é buscar o livro pelo código lendo o arquivo "LIVROS" e retornar uma variável de *struct* que contém todos os dados do livro.

3.4.4 Função alterarLivro()

A função serve para alterar os campos desejados de um determinado livro. No código da função `alterarLivro()` aplica-se os seguintes passos:

1) Solicitar ao usuário o código do livro a ser alterado;
2) Buscar o livro pelo código (utilizar a função `buscarLivroPeloCodigo()` e exibir suas informações na tela);
3) Perguntar quais campos o usuário deseja alterar e armazenar as modificações na variável de estrutura;
4) Sobrescrever o registro com as alterações realizadas.

Observa-se, na Figura 3.10, o funcionamento da função `alterarLivro()`:

Figura 3.10 - Alterando um livro

```
# Opcao-> 3

Codigo do Livro a ser Alterado: 7

Codigo: 7
Titulo: Teste TCC
preco: 50.50
Confirma Livro(S/N)? S

Deseja alterar o Titulo(S/N)? N

Deseja alterar o Preco(S/N)? S

Digite o novo preco: 25

Gravar Livro(S/N)? S

Livro alterado com sucesso.
```

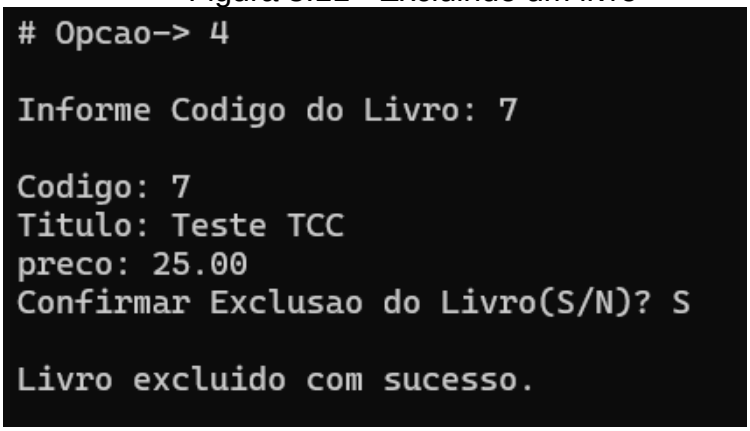
Fonte: Elaborada pelo autor, 2025

3.4.5 Função excluirLivro()

A função utiliza dois arquivos binários: o arquivo original com os livros cadastrados “Livros.dat” (no programa utiliza-se a constante LIVROS no lugar desse nome) e “livrosnew.dat”. Em primeiro lugar precisa-se saber qual livro o usuário deseja excluir. Então, solicita-se ao usuário o código do livro que ele deseja excluir. A função `buscarLivroPeloCodigo()` verifica se o livro foi encontrado. Se sim, suas informações são exibidas na tela e o programa, logo em seguida, pede uma confirmação do usuário para excluir o livro.

Se confirmado, o arquivo “Livros.dat” é aberto para leitura e na sequência é criado o arquivo “livrosnew.dat” no modo escrita. Feito isso, o comando *while* lê registro por registro do arquivo original (“Livros.dat”) e escreve todos os registros no novo arquivo (“livrosnew.dat”) menos o livro com o código que queremos excluir. Por fim, os dois arquivos são fechados, o arquivo antigo é excluído e “livrosnew.dat” é renomeado para “livros.dat”. Verifica-se, na Figura 3.11, a implementação da função excluirLivro():

Figura 3.11 - Excluindo um livro



Fonte: Elaborada pelo autor, 2025

3.4.6 Função relacionarAutorLivro() e relatorioAutoresLivros()

Como mencionado anteriormente, para relacionar livros e autores há um arquivo chamado “RELACAO”. Nele, por meio da *struct reg_RelAutorLivro* é armazenado o código da relação junto do código do livro e código do autor. A função inicia solicitando o código da nova relação, o código do livro e o código do autor. As funções `buscarLivroPeloCodigo()` e `buscarAutorPeloCodigo()` procuram

os registros dos respectivos códigos informados e retornam o registro completo (variável de estrutura). Quando localizados, as informações são exibidas na tela e o programa pede confirmação do usuário para gravar a relação. Se confirmada, o arquivo de relação é aberto para acrescentar dados no seu fim e as informações são gravadas. Observa-se, na Figura 3.12, o registro de uma relação de um livro com um autor:

Figura 3.12 - Relacionando um livro a um autor

```
# Opcao-> 14

Digite o Codigo da relacao autor-livro: 14

Digite o Codigo do Autor: 6

Codigo: 6 - Nome: Eric Ricoldi
Digite o Codigo do Livro: 7

Codigo: 7 - Titulo: Portfolio Eric
Confirma Autor e Livro(S/N)? S

Gravar a relacao Autor-Livro (S/N)? S

Operacao registrada com sucesso.
```

Fonte: Elaborada pelo autor, 2025

A função `relatorioAutoresLivros()` exibe todos os livros e seus respectivos autores. Verifica-se, na Figura 3.13, como aparece a última relação gravada:

Figura 3.13 - Relatório de livros com seus autores

```
# Opcao-> 15

         Relatorio de Livros com Autores

 Codigo Titulo                    Autor                          Preco
 1      Linguagem C               Julio Fontes                    84.53
                                  Carlos Trombini
 2      Estrutura de Dados        Julio Fontes                    94.50
                                  Leandro Carvalho
 3      Redes de Computadores     Maria Senna                     94.50
                                  Marcelo Alves
                                  Eric Ricoldi
                                  Leandro Carvalho
 4      Redes Neurais             Leandro Carvalho               126.00
                                  Cecilia Ventura
 5      Dormindo em Redes         Carlos Trombini                136.50
                                  Cecilia Ventura
 6      HQ Titio Cariane          Eric Ricoldi                    11.50
 7      Portfolio Eric            Eric Ricoldi                     0.00
```

Fonte: Elaborada pelo autor, 2025

Percebe-se que quando um livro possui mais de um autor, o nome do primeiro autor é exibido na mesma linha do nome do livro e os demais autores aparecem logo abaixo sem repetir as informações do livro.

3.4.7 Função efetuarVenda () e relatorioVendasDetalhado()

O registro de vendas segue a mesma lógica da função `relacionarAutorLivro()`. O que muda são os arquivos e *structs* utilizadas. O arquivo “VENDAS” armazena as informações das vendas como: código da venda, código do cliente, código do livro comprado, a quantidade comprada e o desconto aplicado. A função inicia solicitando o código da nova venda, o código do livro comprado e o código do cliente. As funções `buscarLivroPeloCodigo()` e `buscarClientePeloCodigo()` buscam o registro do livro e do cliente informado. Se encontrados, os registros são exibidos na tela. Logo em seguida, o programa questiona se o usuário confirma o cliente e o livro mostrados. Se confirmado, o programa solicita a quantidade de livros comprados e o valor do desconto. Esses valores são armazenados na variável de estrutura venda. Por fim, a função pede a confirmação do usuário para registrar a venda. Se confirmado, o arquivo “VENDAS” é aberto para adicionar o registro da venda. Na Figura 3.14 exibe-se o registro de uma venda:

Figura 3.14 - Registrando uma venda

```
# Opcao-> 18

Digite o Codigo da Venda: 6

Digite o Codigo do Cliente: 4

Codigo: 4 - Nome: Amanda Gabriella
Digite o Codigo do Livro: 4

Codigo: 4 - Titulo: Redes Neurais
Confirma Cliente e Livro(S/N)? S

Digite a Quantidade: 1

Digite o Desconto: 15

Gravar Venda(S/N)? S

Venda registrada com sucesso.
```

Fonte: Elaborada pelo autor, 2025

A função `relatorioVendasDetalhado()` exibe todas as vendas registradas sem repetir informações desnecessárias. Pode-se observar sua saída na Figura 3.15:

Figura 3.15 - listando as vendas de forma otimizada

```
# Opcao-> 21

         Relatorio de Vendas - Detalhado
Venda Cliente                    Livro                      Preco  Desc  Qtde Cobrado
1     2 -Maria Silva             3 -Redes de Computadores   94.50  5.00   1    89.50
2     1 -Ubaldo Ferreira         2 -Estrutura de Dados      94.50  10.00  1    84.50
                                 4 -Redes Neurais           126.00 15.00  1    111.00
3     3 -Joao da Silva           5 -Dormindo em Redes       136.50 30.00  2    243.00
4     2 -Maria Silva             1 -Linguagem C             84.53  15.00  2    154.05
                                 2 -Estrutura de Dados      94.50  0.00   1    94.50
5     1 -Ubaldo Ferreira         6 -HQ Titio Cariane        11.50  0.00   1    11.50
                                 1 -Linguagem C             84.53  10.00  2    159.05
6     4 -Amanda Gabriella        4 -Redes Neurais           126.00 15.00  1    111.00
```

Fonte: Elaborada pelo autor, 2025

3.5 CONCLUSÃO

O conhecimento das estruturas de dados é fundamental para a formação de profissionais capazes de propor e desenvolver soluções com boa otimização, eficientes e escaláveis. O desenvolvimento do sistema da livraria, na linguagem C, utilizando arquivos e *structs* foi de grande valia para o estudante. Visto que, precisou-se compreender realmente como funciona cada uma dessas estruturas, além disso, trabalhou o senso crítico e a lógica do discente para a criação de diferentes funções.

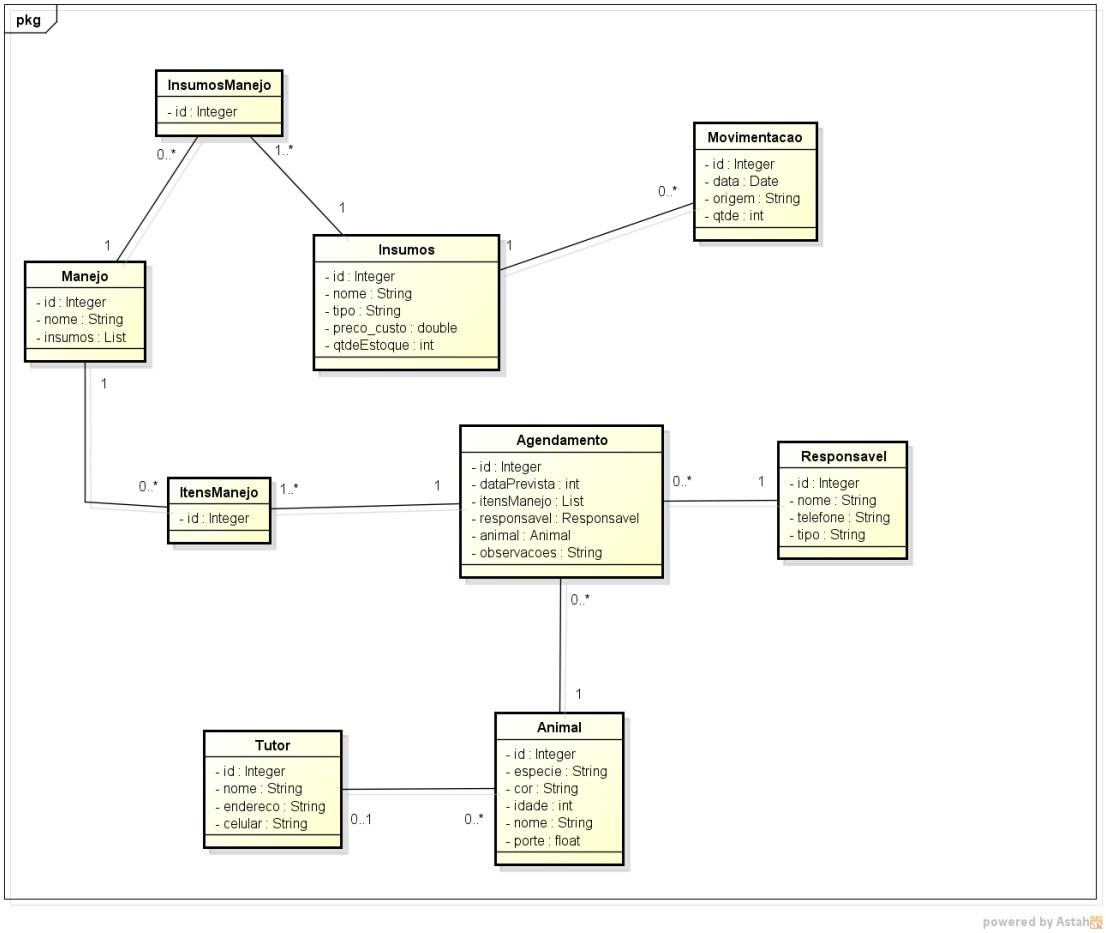
4 PROJETO DO 4º SEMESTRE: API REST PARA GESTÃO DE CASTRAÇÕES DE ANIMAIS

4.1 CONTEXTUALIZAÇÃO E OBJETIVO DA API

> *Application Programming Interface* que, traduzida para o português, pode ser compreendida como uma interface de programação de aplicação. Ou seja, API é um conjunto de normas que possibilita a comunicação entre plataformas por meio de uma série de padrões e protocolos (Fabro, 2020).

As APIs permitem a integração e comunicação de *softwares* com outras plataformas. Sua função é simplificar e facilitar o trabalho dos desenvolvedores. Um exemplo é a utilização da API do Google *Maps* por aplicativos com serviço de mapa (Fabro, 2020). Na disciplina de programação *web*, lecionada pelo professor e coordenador Anderson Pazin, desenvolveu-se uma API *Rest* voltada à gestão de castração de animais. Tratou-se de um projeto extensionista que envolveu a matéria de programação *web* e a matéria de banco de dados ministrada pelo professor Luiz Fernando. O objetivo do sistema era servir de apoio às atividades realizadas por uma organização não governamental (ONG) da cidade de Avanhandava, onde, realiza-se castrações de animais em condições de abandono. O projeto foi desenvolvido na linguagem Java com uso dos *frameworks* Spring e Hibernate. O sistema foi estruturado sob a arquitetura *Model-View-Controller* (MVC). Segundo Mak (2008), a arquitetura MVC divide a aplicação em três camadas: *model* (executa regras de negócio, processa as informações e persiste as entidades no banco de dados); *view* (interface que interage com o usuário); *controller* é o intermediador entre a *view* e a *model*, garante comunicação organizada e mantém duas camadas independentes. Verifica-se, na Figura 4.1, o modelo de entidade e relacionamento (MER) da API:

Figura 4.1 - MER da API



Fonte: Elaborada pelo autor, 2025

A principal função da API é a gestão das castrações e outros tipos de manejo por meio de agendamentos. Um agendamento envolve um responsável (um veterinário ou um voluntário), o animal que será o paciente e qual ou quais manejos serão necessários. Além da castração existe outros manejos como: banho, tosa, medicação e aplicação de curativos. Para cada tipo de manejo há insumos necessários. Os insumos são os utensílios veterinários e os demais produtos que são utilizados pelos manejos. Até o momento a entidade movimentação apenas registra as entradas de insumos que podem ser de origem diferentes como: doação ou compra. Uma melhoria que pode ser implementada futuramente nessa API é a administração dos insumos, ou seja, quando chegar insumos por meio das movimentações e a medida em que os recursos são consumidos, refletir as mudanças na entidade Insumos.

4.2 TECNOLOGIAS UTILIZADAS NO DESENVOLVIMENTO DA API

4.2.1 Conceito de API Rest

A interação da API com os clientes é por meio de *web services* (serviços web) – mais especificamente o protocolo *Hypertext Transfer Protocol* (HTTP) e suas requisições. Nesse sentido, é importante conhecer as seguintes siglas: JavaScript *Object Notation* (JSON); *Extensible Markup Language* (XML); *Uniform Resource Locator* (URL). Consoante a Augusto (2019), *web services,* trata-se de:

> Uma tecnologia que permite a comunicação entre aplicações de maneira independente de linguagem de programação e de sistema operacional[...] Os *Web Services* são componentes que permite que as aplicações enviem e recebam dados geralmente em formato XML ou JSON.

Dessa forma, “quando um cliente faz uma requisição, é retornado uma resposta no formato XML ou JSON, quando a API precisa de alguma informação do cliente, o formato enviado também precisa ser o mesmo” (Junior; Rocha; Maciel, 2021, p. 502).

Uma API *Rest* é “baseada em um conjunto de princípios que definem como os recursos de uma aplicação devem ser expostos através de URLs e como esses recursos podem ser manipulados por métodos HTTP” (Silva, 2023). Há alguns princípios básicos para a construção de uma API *Rest*: em primeiro lugar é realizar a separação entre a camada cliente e a camada servidor; o segundo princípio diz que a arquitetura deve permitir a inclusão de camadas intermediárias entre cliente e servidor (*gateways*, *caches* e *load balancers*); Já o terceiro princípio exige que a interface forneça um conjunto de operações básicas que permitam realizar CRUDs por meio de métodos HTTP como *GET*, *POST*, *PATCH*, *PUT* e *DELETE* (Silva, 2023).

4.2.2 Spring e Spring Boot

“O Spring é um *framework* para desenvolvimento de aplicações em Java. Ele facilita a criação de aplicações robustas e de alta qualidade, ajudando os desenvolvedores a lidarem com a complexidade de grandes projetos” (Alves, 2024). O Spring oferece recursos que facilitam o processo de desenvolvimento de *softwares* como: Spring Core, Spring MVC, Spring Data, Spring Cloud e o Spring Boot. “O Spring

Core é a parte principal do Spring *Framework* [...] simplifica a gestão das dependências entre os objetos da aplicação” (Alves, 2024).

O Spring Boot simplifica o processo de configuração e desenvolvimento de aplicativos Java. Essa ferramenta “fornece um conjunto de padrões e convenções predefinidos para ajudar os desenvolvedores a começarem rapidamente sem a necessidade de configurar manualmente muitos aspectos da aplicação” (Alves, 2024). O Spring Boot automatiza tarefas como: configuração de servidor, gestão de dependências e inicialização de aplicativos. Permitindo que o desenvolvedor se concentre mais nas regras de negócio da aplicação. Para implementar o Spring Boot nos projetos pode-se utilizar uma ferramenta de gerenciamento de dependências como o Maven ou Gradle (Alves, 2024).

O Apache Maven é uma ferramenta de automação de compilação amplamente utilizada no desenvolvimento de software Java. Ele simplifica e gerencia o processo de compilação, teste e empacotamento de um projeto Java. Ele encontra e organiza as "peças" necessárias para o projeto (chamadas de dependências), seguindo receitas padrão (Alves, 2024).

4.2.3 Hibernate

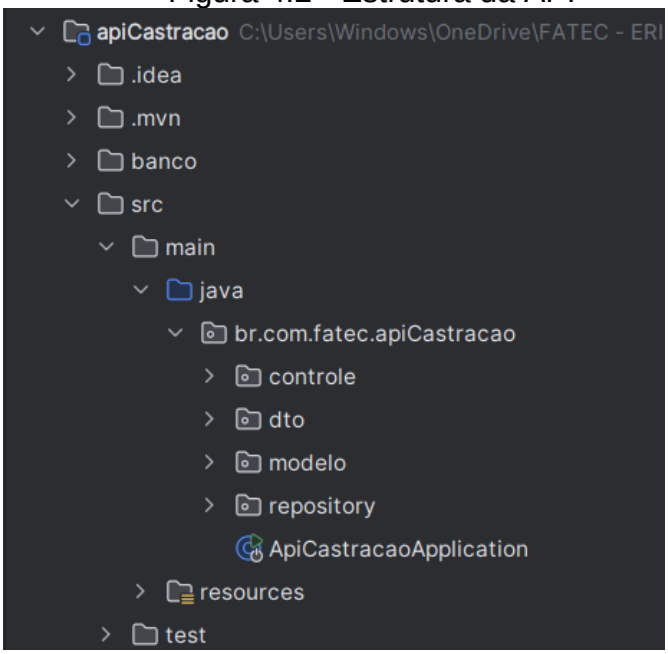
De acordo com site Rocketseat “o Hibernate é um *framework* de mapeamento objeto-relacional (ORM) para Java que atua como um intermediário entre objetos da linguagem e tabelas de banco de dados”. O *framework* elimina a necessidade de escrever consultas SQL complexas; oferece maior portabilidade, pois, o mesmo código pode ser usado com diferentes bancos de dados; alterações no banco de dados reflete facilmente na aplicação; proporciona, também, segurança e maior produtividade. “O Hibernate utiliza uma combinação de anotações e/ou arquivos de configuração XML para mapear as classes Java às tabelas do banco de dados” (Rocketseat, 2025).

4.3 ESTRUTURA DA API

O projeto baseado na arquitetura MVC divide a aplicação em 4 pacotes: controle, dto, modelo e *repository.* Essa forma de arquitetura garante modularidade

ao sistema e replicação de código. Para cada uma das entidades da API há uma classe modelo, uma interface *repository,* uma classe *record* e uma classe controle. Pode-se observar a organização do sistema na Figura 4.2:

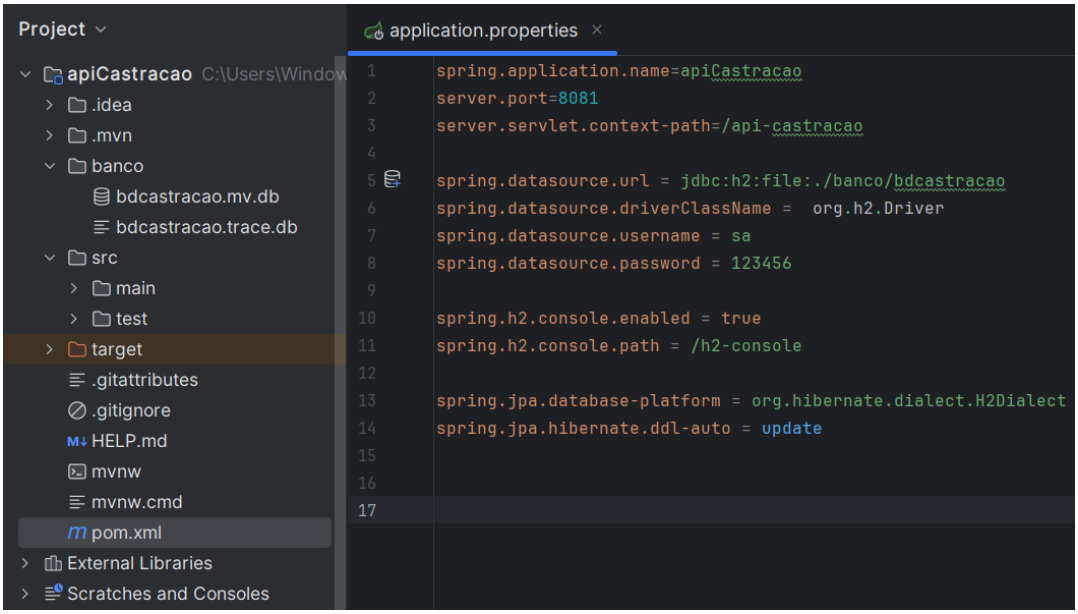
Figura 4.2 - Estrutura da API



Fonte: Elaborada pelo autor, 2025

Para o funcionamento do sistema são necessários outros arquivos como o `pom.xml`, que segundo Alves (2024) é onde devemos adicionar as dependências do Spring ou Spring Boot. Conforme Junior, Rocha e Maciel (2021) o *framework* Spring Boot simplifica configurações por meio do arquivo `application.properties` ou `application.yml`. Verifica-se, na Figura 4.3, a presença do arquivo de dependências (`pom.xml`) e o arquivo de configuração do banco de dados (`application.properties`):

Figura 4.3 - Arquivos de configuração

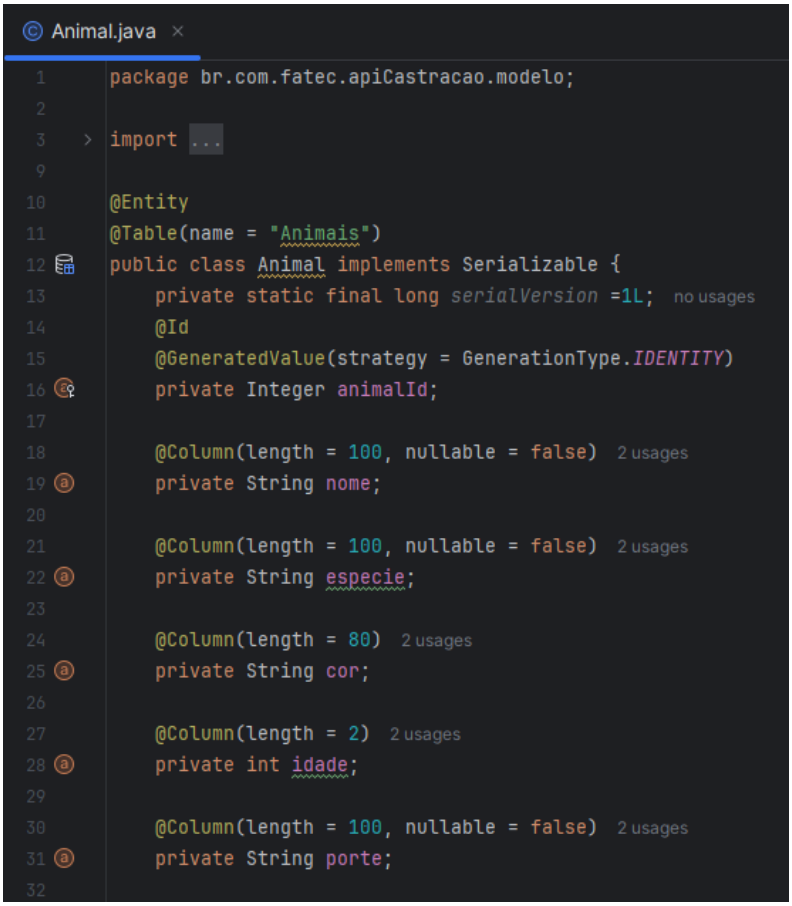


Fonte: Elaborada pelo autor, 2025

4.3.1 Classes Modelo

As classes modelos, através do Hibernate, definem as entidades que serão persistidas no banco de dados. Ao rodar a aplicação ocorre o mapeamento da classe, ou seja, a classe torna-se uma tabela no banco de dados e seus atributos são transformados em colunas da tabela. A Figura 4.4 exibe a classe modelo Animal:

Figura 4.4 - Classe modelo: animal



Fonte: Elaborada pelo autor, 2025

Nota-se que a classe Animal possui um ID (identificador). Esse campo será sua chave primária no banco de dados. Os demais atributos: nome, espécie, cor, idade e porte formam as outras colunas da tabela Animal. Conforme dito anteriormente, o Hibernate possibilita criar relacionamentos entre entidades. Na API, a classe Animal se relaciona com outras duas: Tutor e Agendamento. A regra de negócio diz que um Tutor pode ter muitos animais e um animal só pode pertencer a um único Tutor. Já o relacionamento entre Animal e Agendamento é o seguinte: um animal pode estar associado a muitos agendamentos ao longo do tempo, no entanto, um agendamento deve conter apenas 1 (um) animal. Observa-se, na Figura 4.5, as anotações que realizam essas associações entre as entidades:

Figura 4.5 - Anotações de Relacionamentos

```
33      public Animal(){
34
35      }
36
37      //relacionamento de Animal com tutor
38      @ManyToOne  2 usages
39      @JoinColumn(name = "Tutor")
40      @JsonBackReference
41      private Tutor tutor;
42
43      //relacionamento de Animal e Agendamento
44      @OneToMany(mappedBy = "animal", cascade = CascadeType.ALL)  2 usages
45      @JsonManagedReference
46      private List<Agendamento> agendamentos;
47
48      //métodos acessores
```

Fonte: Elaborada pelo autor, 2025

`@ManyToOne` - significa muitos para um. Nesse caso, muitos animais pertencem a um tutor. Ao declarar um objeto Tutor como atributo de animal, o Hibernate cria um campo na tabela animal que armazena o ID do tutor. Ou seja, a entidade animal recebe a chave primária de tutor, formando assim, uma chave estrangeira. A anotação que relaciona animal com agendamento é `@OneToMany` (um para muitos). Essa anotação cria um relacionamento 1:N, o trecho de código: `mappedBy = "animal", cascade = CascadeType.ALL`, indica que na entidade agendamento há um campo mapeado chamado animal e que quando alterações são realizadas na entidade animal ocorrerá alteração também na entidade agendamento. Isso significa que se um animal for salvo, deletado ou atualizado, isso será aplicado automaticamente aos registros de agendamentos que contenham o ID desse animal.

4.3.2 Classes DTO

As classes *Record* são utilizadas para criar *Data Transfer Object* (DTO), em tradução para o português significa objeto de transferência de dados. Verifica-se, na Figura 4.6, como os registros de animais são criados com base no DTO.

Figura 4.6 - AnimalDTO

```java
package br.com.fatec.apiCastracao.dto;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;


public record AnimalDTO(@NotBlank String nome, @NotNull String especie,  4 usages
                        String cor, int idade, String porte, Integer tutorId) {
}
```

Fonte: Elaborada pelo autor, 2025

4.3.3 Interfaces Repository

Os registros são salvos no banco de dados via *repository.* Tratam-se de interfaces com diversos métodos prontos para salvar recursos, exibir um recurso por seu ID, exibir uma lista de recursos, atualizar e excluir um recurso. Para cada entidade do projeto há uma interface *repository*. Observa-se o repositório da classe Animal na Figura 4.7:

Figura 4.7 - Interface Repositório

```java
package br.com.fatec.apiCastracao.repository;

import br.com.fatec.apiCastracao.modelo.Animal;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AnimalRepository extends JpaRepository<Animal, Integer> {  4 usages
}
```

Fonte: Elaborada pelo autor, 2025

4.3.4 Controllers

As classes *controllers* são controladores *rest* que lidam com requisições HTTP e retorna respostas diretamente (normalmente JSON). Observa-se, na Figura 4.8, a classe AnimalController que permite CRUDs de registros de animais por meio de *endpoints.*

Figura 4.8 - AnimalController

```java
@RestController
public class AnimalController {
    @Autowired
    AnimalRepository repositorioAnimal;

    @Autowired
    TutorRepository repositorioTutor;

    @PostMapping("/animais")
    public ResponseEntity<Animal> salvarAnimal(@RequestBody AnimalDTO animalDTO){
        var animalModelo = new Animal();
        BeanUtils.copyProperties(animalDTO, animalModelo, ...ignoreProperties: "tutorId");

        if (animalDTO.tutorId() != null) { // Converte tutorId -> Tutor
            Tutor tutor = repositorioTutor.findById(animalDTO.tutorId())
                    .orElseThrow(() -> new RuntimeException("Tutor não encontrado"));
            animalModelo.setTutor(tutor);
        }

        return ResponseEntity
                .status(HttpStatus.CREATED)
                .body(repositorioAnimal.save(animalModelo));
    }
```

Fonte: Elaborada pelo autor, 2025

Pode-se observar a implementação do método *POST* que cria um novo registro de Animal (operação de criação).

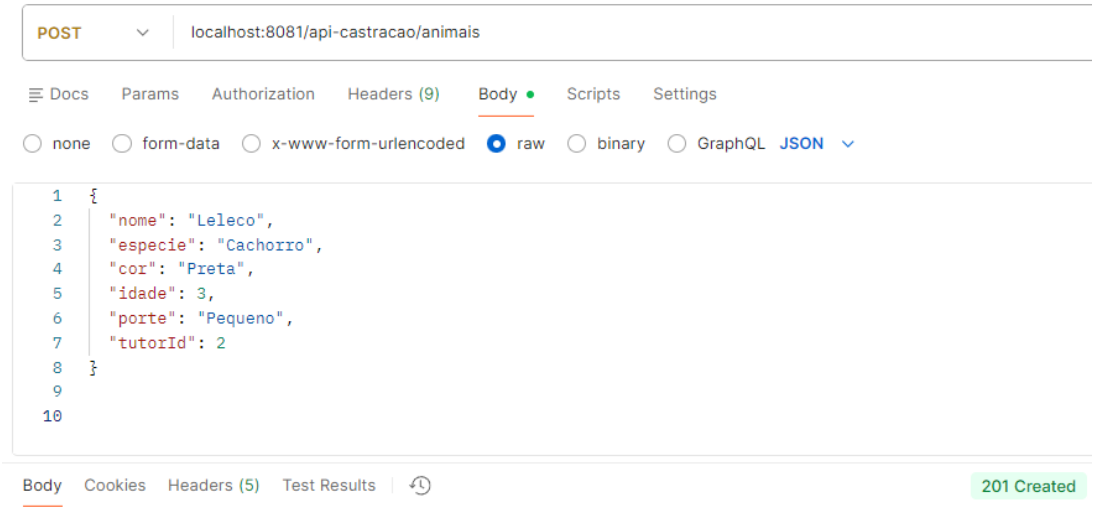
4.4 TESTES NA API

Para a realização de testes de CRUD utilizou-se o *software* Postman que permite o envio e recebimento de requisições HTTP no formato JSON.

4.4.1 Método Post

A Figura 4.9 exibe uma requisição do tipo *POST* para a criação de um novo registro de animal. Percebe-se que, no JSON, há os atributos de animais e também o “tutorId”, ou seja, qual o tutor desse animal.

Figura 4.9 - Método *Post*

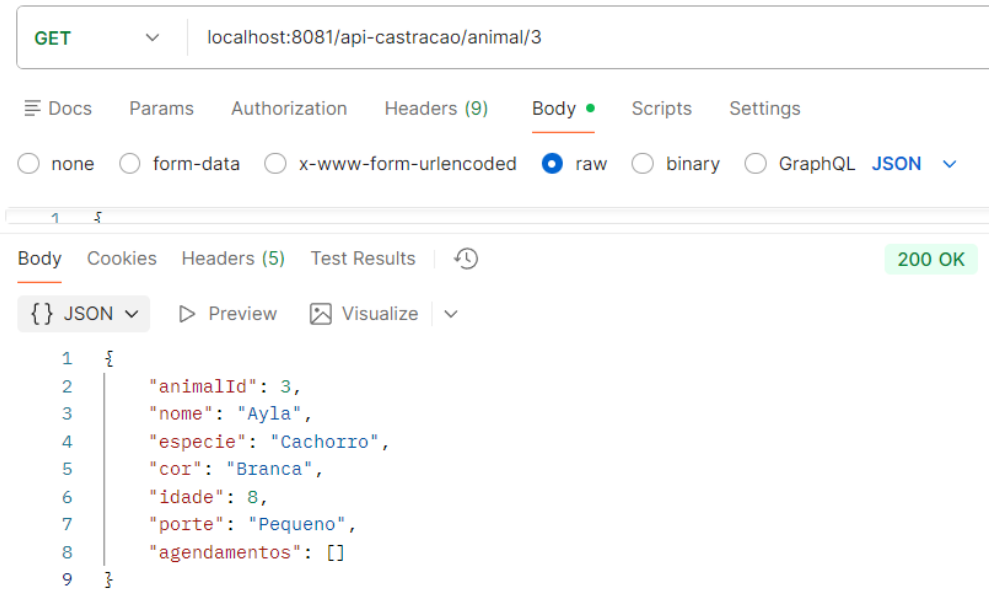


Fonte: Elaborada pelo autor, 2025

4.4.2 Método Get

O método *GET* retorna um registro do banco de dados. No exemplo da Figura 4.10 requisitou-se o registro do animal de ID igual a 3.

Figura 4.10 - Método *Get*

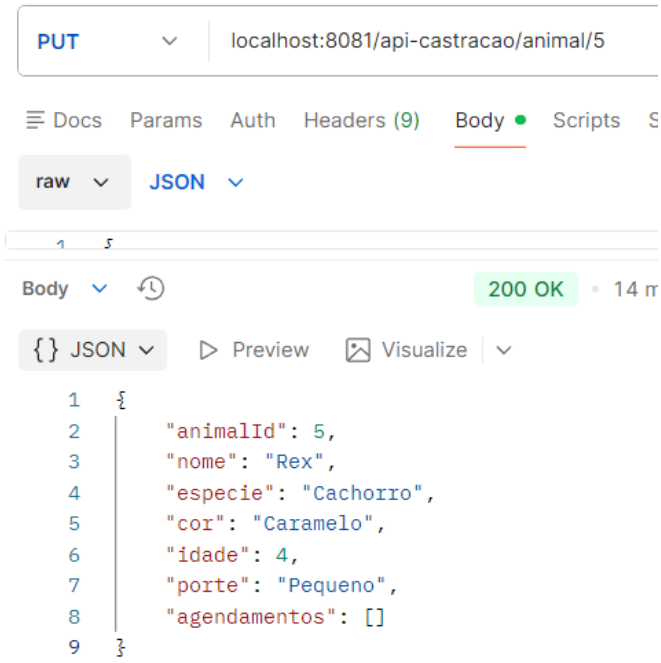


Fonte: Elaborada pelo autor, 2025

4.4.3 Método Put

Esse método serve para atualizar um registro existente. Observa-se um exemplo na Figura 4.11 onde o registro de ID igual a 5 sofreu alterações:

Figura 4.11 - Método *Put*

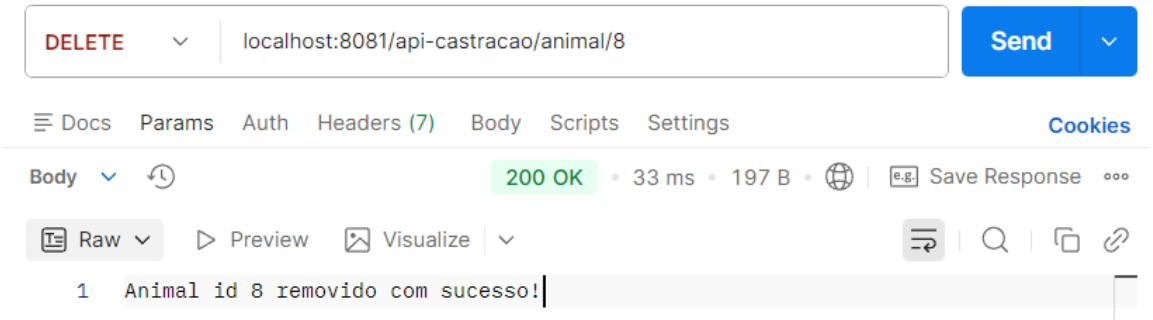


Fonte: Elaborada pelo autor, 2025

4.4.4 Método Delete

Para excluir um registro basta utilizar o método *DELETE*. Pode-se observar na Figura 4.12 que para excluir um registro basta informar qual o ID na URL.

Figura 4.12 - Método *Delete*



Fonte: Elaborada pelo autor, 2025

4.4.5 Selects no banco de dados

Para testes na API, além do Postman, pode-se realizar consultas diretamente do banco de dados da API. Essa forma de teste serve para verificar se as informações estão sendo persistidas no banco de forma correta. É possível observar, também, se os relacionamentos entre as entidades estão funcionando conforme o esperado.

Verifica-se, na Figura 4.13, um exemplo de consulta que envolve as entidades animais e tutor:

Figura 4.13 - Select Animais e Tutores

```
SELECT a.animal_id, a.nome AS Nome_Animal, t.nome AS Nome_Tutor
from animais a, tutor t
where a.tutor = t.tutor_id;
```

ANIMAL_ID	NOME_ANIMAL	NOME_TUTOR
1	Mel	Renata Furlan
2	Bob	Marcelo Alves
3	Ayla	Gabriella Fernanda
4	Thor	Silvio Luiz
5	Rex	Silvio Luiz
6	Lola	Maicon Douglas
7	Leleco	Silvio Luiz

(7 rows, 2 ms)

Fonte: Elaborada pelo autor, 2025

Essa consulta mostra o ID e nome do Animal seguido do nome do seu Tutor. Pode-se observar por esse *select* simples, por exemplo, que o tutor de nome “Silvio Luiz” é dono de três animais.

Agora, pode-se observar, na Figura 4.14, uma consulta com um grau um pouco maior de complexidade que envolve mais de três tabelas. Esse select exibe o ID dos agendamentos, a data do agendamento, o nome do responsável pelo agendamento, qual o animal e o manejo que será realizado. O agendamento de ID igual a 1, por exemplo, está marcado para o dia 10/12/2025, a responsável é a Camila Dias, o animal é a Ayla e será realizado dois manejos: castração e medicação.

Figura 4.14 - Select Agendamentos

```
SELECT
   A.AGENDAMENTO_ID,
   A.DATA_PREVISTA,
   R.NOME AS NOME_RESPONSAVEL,
   AN.NOME AS NOME_ANIMAL,
   M.NOME AS NOME_MANEJO
FROM
   AGENDAMENTOS A
JOIN
   ANIMAIS AN ON A.ANIMAL = AN.ANIMAL_ID
JOIN
   RESPONSAVEL R ON A.RESPONSAVEL = R.RESPONSAVEL_ID
JOIN
   ITENS_MANEJO IM ON A.AGENDAMENTO_ID = IM.AGENDAMENTO
JOIN
   MANEJO M ON IM.MANEJO = M.MANEJO_ID
ORDER BY
   A.AGENDAMENTO_ID, M.NOME;
```

AGENDAMENTO_ID	DATA_PREVISTA	NOME_RESPONSAVEL	NOME_ANIMAL	NOME_MANEJO
1	2025-12-10 05:30:00	Camila Dias	Ayla	Castração
1	2025-12-10 05:30:00	Camila Dias	Ayla	Medicação
2	2025-12-11 06:00:00	Josué Batista	Mel	Medicação
3	2025-12-11 21:00:00	Kayo Ribeiro	Thor	Banho
3	2025-12-11 21:00:00	Kayo Ribeiro	Thor	Tosa

(5 rows, 9 ms)

Fonte: Elaborada pelo autor, 2025

4.5 CONCLUSÃO

O desenvolvimento de uma API *Rest* na linguagem Java foi um projeto muito proveitoso aos discentes, pois, foi desenvolvido um sistema sob uma arquitetura e utilizando *frameworks* do mercado de trabalho. O projeto baseado na arquitetura MVC apresentou vantagens como: modularidade e aproveitamento de código. O uso dos *frameworks* Spring e Hibernate contribuiu para um desenvolvimento acelerado e descomplicado, visto que eles automatizam etapas que exigiriam mais esforço e tempo para implementar.

5 PROJETO DO 5º SEMESTRE: COMANDOS SQL DE CONSULTA A UM BANCO DE DADOS

5.1 CONTEXTUALIZAÇÃO E OBJETIVO

"O SQL é a sigla para *Structured Query Language*, que em português significa Linguagem de consulta estruturada, é uma linguagem padrão para trabalhar com bancos de dados relacionais" (Oliveira; Dias, 2019). "A linguagem SQL é relativamente semelhante entre os principais Sistemas Gerenciadores de Banco de Dados (SGBDs) do mercado, como: Oracle, MySQL, MariaDB, PostgreSQL, Microsoft SQL Server, entre outros" (Oliveira; Dias, 2019).

Oliveira e Dias (2019) explicam que o SQL é amplamente utilizado por cientistas de dados e profissionais que utilizam Excel. E não para por aí, o SQL está presente em diversas outras áreas como: no desenvolvimento de aplicativos *web* que utilizam banco de dados; ciência de dados; educação e pesquisa; gestão de finanças e contabilidade. Os dados na sociedade atual são fundamentais, visto que, todos nós consumimos e produzimos dados. "Assim, em diferentes contextos, da economia, da ciência, da saúde e da educação, a coleta e a análise de informações são pontos fundamentais para embasar decisões criteriosas e promover avanços significativos" (Oliveira; Dias, 2019).

Os comandos SQL são categorizados em várias linguagens específicas. Anjos (2023) descreve cada uma:

- **DDL** (*Data Definition Language*) - Linguagem de Definição de Dados. São os comandos que interagem com os objetos do banco. São comandos DDL: *CREATE*, *ALTER* e *DROP*;
- **DML** (*Data Manipulation Language*) - Linguagem de Manipulação de Dados. São os comandos que interagem com os dados dentro das tabelas. São comandos DML: *INSERT*, *DELETE* e *UPDATE*;
- **DQL** (*Data Query Language*) - Linguagem de Consulta de dados. São os comandos de consulta. *SELECT* é o comando de consulta;
- **DTL** (*Data Transaction Language*) - Linguagem de Transação de Dados. São os comandos para controle de transação. São comandos DTL: *BEGIN TRANSACTION, COMMIT E ROLLBACK*;

- **DCL** (*Data Control Language*) - Linguagem de Controle de Dados. São os comandos para controlar a parte de segurança do banco de dados. São comandos DCL: *GRANT*, *REVOKE* E *DENY*.

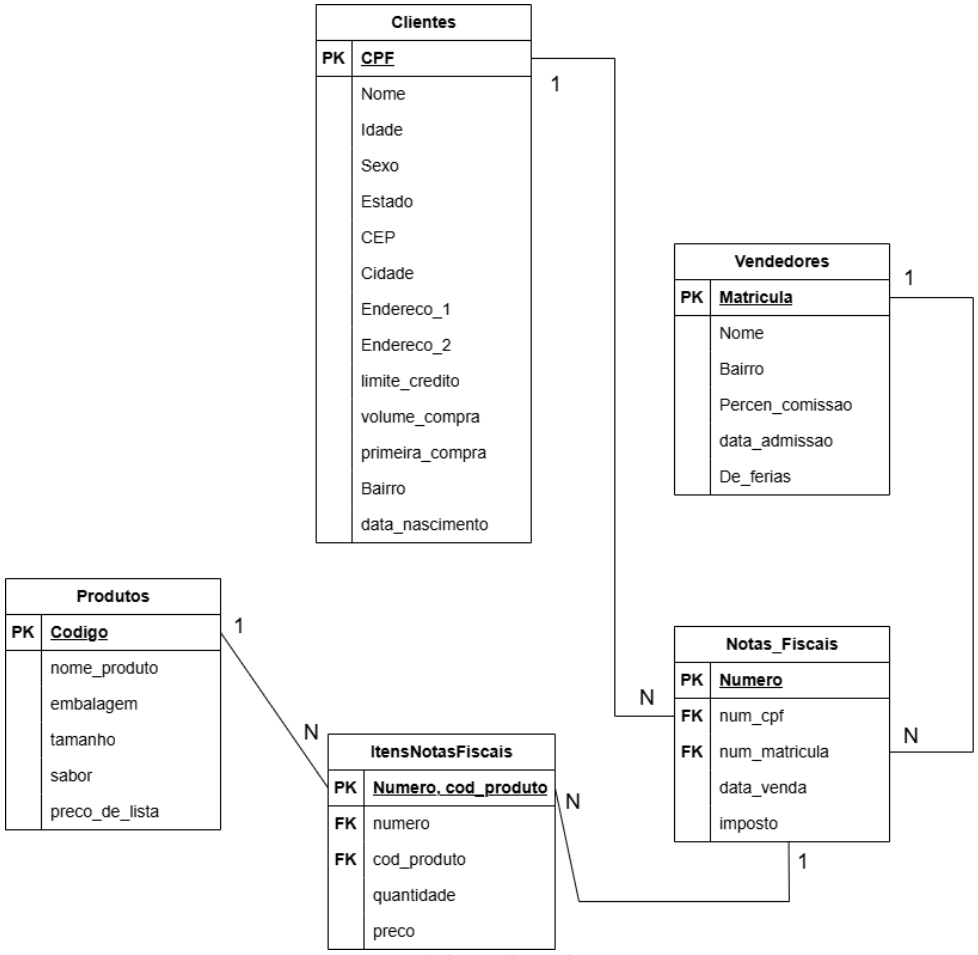
Na disciplina de laboratório de banco de dados ministrada pelo professor Thiago Bergamaschi ensinou-se, aos discentes, comandos SQL fundamentais utilizando o SGBD da Oracle: SQL Developer. Na disciplina, aprendeu-se, criar bancos de dados, criar tabelas, realizar alterações, inserções, atualizações e exclusão de dados. O foco maior das aulas, no entanto, foi na linguagem DQL, isto é, a linguagem de consultas de dados. Sendo assim, praticou-se nas aulas, comandos de consulta em um banco de dados criado em aula. O objetivo do projeto do 5º semestre é apresentar e explicar comandos SQL úteis de consulta aos dados.

5.2 BANCO DE DADOS UTILIZADO NAS CONSULTAS

5.2.1 Banco de dados “Notas Fiscais”

Como o próprio nome sugere, esse banco de dados registra notas fiscais de compras de produtos. Pode-se observar, na Figura 5.1, o MER do banco de dados.

Figura 5.1 - MER do banco de dados



Fonte: Elaborada pelo autor, 2026

A estrutura é composta por 5 tabelas: Vendedores, Clientes, Produtos, Notas_Fiscais e ItensNotasFiscais. A tabela Notas_Fiscais armazena o número da nota fiscal, o código do vendedor (quem atendeu o cliente), o código do cliente (quem comprou), a data da venda e o imposto. A tabela ItensNotasFiscais é uma tabela intermediária que resolve o problema de um relacionamento muitos-para-muitos entre as entidades Notas_Fiscais e Produtos.

Dessa forma, a entidade ItensNotasFiscais persiste no banco de dados as informações dos itens comprados e a quantidade. Na tentativa de montar o relacionamento de Produtos e Notas_Fiscais sem uma tabela intermediária causaria sérios problemas como: redundância, desperdício e inconsistência de dados. Isso ocorreria, pois, seria necessário repetir a nota fiscal para cada produto ou então repetir produtos dentro da nota (campos produto1, produto2…).

A chave primária da tabela ItensNotasFiscais é composta pelo número da nota fiscal e o código do produto. Essa estrutura, portanto, permite armazenar, sem

problemas, as informações dos itens da nota fiscal. A Figura 5.2 é um exemplo de como funciona as tabelas Notas_Fiscais e ItensNotasFiscais.

Figura 5.2 - Exemplo tabelas

Tabela Notas_Fiscais

id_nota	id_cliente	id_vendedor	data
100	1	1	01/03/2026
101	2	2	01/03/2026

Tabela ItensNotasFiscais

id_nota	id_produto	quantidade	valor_unitario
100	1	2	R$ 20,00
100	2	1	R$ 10,00
101	3	3	R$ 8,00
101	1	2	R$ 10,00

Isso representa, por exemplo:

Nota 100
Cliente: João
Vendedor: Carlos
Itens:
→ 2 Arroz
→ 1 Feijão

Nota 101
Cliente: Maria
Vendedor: Ana
Itens:
→ 3 Óleo de Soja
→ 2 Arroz

Fonte: Elaborada pelo autor, 2026

5.2.2 Chave primária x chave estrangeira

Chave primária, ou *Primary Key* (PK), é o identificador único da tabela, sendo representada por um ou mais campos que não podem receber valores repetidos. Cada registro da tabela deve possuir uma, e somente uma, chave primária; chaves primárias não podem ser nulas e normalmente são implementadas automaticamente pelo banco de dados (Machado, 2020). Machado (2020), completa: “são as chaves para o relacionamento entre entidades ou tabelas da base de dados. Assim haverá na tabela relacionada uma referência a essa chave primária (que será, na tabela relacionada, a chave estrangeira)”.

A chave estrangeira, ou *Foreign Key* (FK), trata-se de um campo em uma tabela que faz referência a chave primária de outra tabela. Diferente da PK, a FK pode ser nula e é possível ter mais de uma (ou nenhuma) em uma tabela (Machado, 2020).

A Figura 5.3 exibe o comando SQL de criação da tabela vendedores do banco de dados “notas fiscais”.

Figura 5.3 - Criação da tabela vendedores

```
Create table Vendedores (
Matricula varchar2(5),
Nome varchar2(100),
Percen_comissao float,
data_admissao date,
De_ferias number (1),
Bairro varchar2(50),
constraint pk_vendedor PRIMARY KEY (Matricula)
);
```

Fonte: Elaborada pelo autor, 2026

Nota-se que a tabela possui o campo “Matricula” como chave primária. O comando `constraint` no SQL define as regras/restrições de uma coluna de uma tabela do banco de dados. “Podemos dizer, também, que representam propriedades que os dados de certa coluna precisam obedecer. Chaves primárias e chaves estrangeiras são exemplos de *constraints*” (Machado, 2020). Pode-se observar, na Figura 5.4, um exemplo da utilização de chave estrangeira.

Figura 5.4 - Criação da tabela notas_fiscais

```
create table Notas_fiscais (
Numero int,
num_cpf varchar2 (11),
num_matricula varchar2 (5),
data_venda date,
imposto float,
constraint pk_nota_fiscal PRIMARY KEY (Numero),
constraint fk_notafiscal_vendedor FOREIGN KEY (num_matricula)
references Vendedores (Matricula),
constraint fk_notafiscal_cliente FOREIGN KEY (num_cpf)
references Clientes (CPF)
);
```

Fonte: Elaborada pelo autor, 2026

O comando em questão cria a tabela Notas_Fiscais que possui como chave primária o campo “numero”. Sabe-se que essa tabela armazena o cliente e o vendedor da compra. Para isso, deve haver, na entidade Notas_Fiscais referência à chave primária da entidade clientes e da entidade vendedores. Portanto, a entidade Notas_Fiscais possui duas chaves estrangeiras: o campo “num_cpf” (referência a PK da tabela clientes) e o campo “num_matricula” (referência a PK da tabela vendedores).

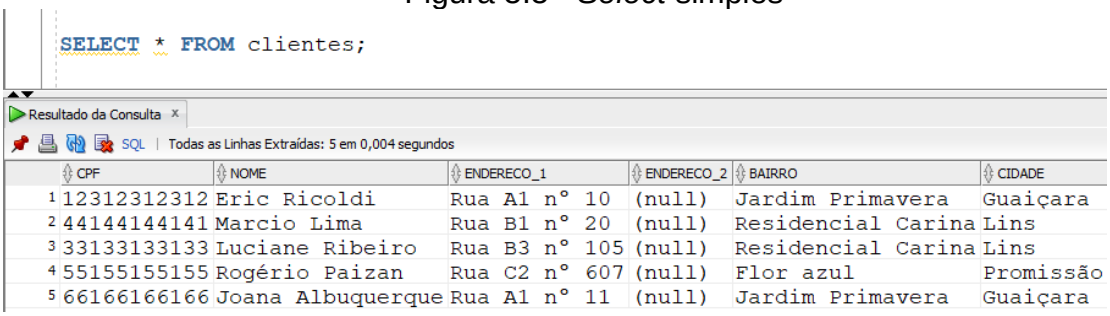
5.3 COMANDOS SELECT

A estrutura básica do *SELECT* envolve selecionar as colunas desejadas de uma ou mais tabelas, podendo incluir filtros e ordenações: `SELECT colunas FROM tabela [comandos adicionais];` (Monutti, 2024).

5.3.1 Select simples

Observa-se, na Figura 5.5, o seguinte comando *select*:

Figura 5.5 - *Select* simples



	CPF	NOME	ENDERECO_1	ENDERECO_2	BAIRRO	CIDADE
1	12312312312	Eric Ricoldi	Rua A1 nº 10	(null)	Jardim Primavera	Guaiçara
2	44144144141	Marcio Lima	Rua B1 nº 20	(null)	Residencial Carina	Lins
3	33133133133	Luciane Ribeiro	Rua B3 nº 105	(null)	Residencial Carina	Lins
4	55155155155	Rogério Paizan	Rua C2 nº 607	(null)	Flor azul	Promissão
5	66166166166	Joana Albuquerque	Rua A1 nº 11	(null)	Jardim Primavera	Guaiçara

Fonte: Elaborada pelo autor, 2026

Esse comando exibe todos os dados da tabela clientes. A presença do asterisco faz com que seja selecionado todas as colunas da tabela. Observação: para melhor visualização do resultado do *select,* a Figura 5.5 foi recortada. Sendo assim, no *print* não é exibido todas as colunas do resultado da consulta.

Na Figura 5.6 é mostrado um comando *select* que indica quais as colunas devem ser exibidas:

Figura 5.6 - *Select* de colunas específicas

```
SELECT nome, cidade
FROM clientes;
```

Resultado da Consulta

SQL | Todas as Linhas Extraídas: 5 em 0,043 segundos

	NOME	CIDADE
1	Eric Ricoldi	Guaiçara
2	Marcio Lima	Lins
3	Luciane Ribeiro	Lins
4	Rogério Paizan	Promissão
5	Joana Albuquerque	Guaiçara

Fonte: Elaborada pelo autor, 2026

5.3.2 Select com filtro

"O comando *SELECT* permite adicionar condições para filtrar os dados usando a cláusula `WHERE`. Isso é útil quando você deseja recuperar apenas registros que atendem a certos critérios" (Monutti, 2024). Observa-se um exemplo na Figura 5.7:

Figura 5.7 - *Select* com filtro

```
SELECT nome, cidade, idade
from clientes
WHERE sexo = 'F';
```

Resultado da Consulta

SQL | Todas as Linhas Extraídas: 2 em 0,003 segundos

	NOME	CIDADE	IDADE
1	Luciane Ribeiro	Lins	46
2	Joana Albuquerque	Guaiçara	33

Fonte: Elaborada pelo autor, 2026

Essa consulta exibiu o nome, a cidade e a idade dos clientes do sexo feminino. Pode-se realizar consultas com múltiplos filtros. A Figura 5.8 é um exemplo:

Figura 5.8 - *Select* com mais de um filtro

```
select nome, cidade, idade, limite_credito as Limite_de_Credito
from clientes
where cidade = 'Lins' and limite_credito > 12000;
```

Resultado da Consulta

Todas as Linhas Extraídas: 1 em 0,003 segundos

	NOME	CIDADE	IDADE	LIMITE_DE_CREDITO
1	Luciane Ribeiro	Lins	46	15000

Fonte: Elaborada pelo autor, 2026

Essa consulta possui dois filtros: exibir apenas os clientes da cidade de Lins e que tenham limite de crédito superior a R$ 12.000,00. O operador `and` faz toda a diferença, pois, só serão exibidos os registros que atendam, obrigatoriamente, os dois critérios. O trecho: `limite_credito as Limite_de_Credito`, faz com que a coluna `limite_credito` seja exibida com o nome “Limite_de_Credito”.

5.3.3 Select envolvendo mais de uma tabela

O comando select da Figura 5.9 lista o número da nota fiscal, o CPF do cliente, o nome do vendedor e a data da venda.

Figura 5.9 - Consulta envolvendo duas tabelas

```
SELECT nf.numero AS Numero_Nota_Fiscal, nf.num_cpf AS CPF_Cliente,
v.nome AS Nome_Vendedor, nf.data_venda
FROM notas_fiscais nf, vendedores v
WHERE nf.num_matricula = v.matricula;
```

Resultado da Consulta

Todas as Linhas Extraídas: 8 em 0,004 segundos

	NUMERO_NOTA_FISCAL	CPF_CLIENTE	NOME_VENDEDOR	DATA_VENDA
1	1000	12312312312	Bianca Farias	06/02/26
2	1001	66166166166	Bianca Farias	05/02/26
3	1002	44144144141	Luana Campos	06/02/26
4	1003	33133133133	Guilherme Reis	07/02/26
5	1004	55155155155	Alexandre Junior	05/02/26
6	1005	66166166166	Luana Campos	03/02/26
7	1006	12312312312	Luana Campos	07/02/26
8	1007	44144144141	Guilherme Reis	04/02/26

Fonte: Elaborada pelo autor, 2026

Nessa consulta foi utilizado duas tabelas: Vendedores e Notas_Fiscais. Vale lembrar que existe um relacionamento entre essas entidades. Na tabela Notas_Fiscais existe um campo que faz referência à chave primária da tabela vendedores, ou seja, as tabelas estão ligadas por meio da FK “num_matricula”. Isso explica o trecho de código (`nf.num_matricula = v.matricula`). Isto é, os valores da coluna

`num_matricula` devem ser iguais aos valores da coluna chave primária da tabela vendedores (campo matricula). Foi utilizado apelidos no lugar do nome das tabelas. No lugar de escrever “notas_fiscais”, podemos utilizar apenas “nf”, por exemplo. Para isso, basta fazer o que está na linha do `from` – especificar qual o apelido da tabela.

5.3.4 Order by e Group by

Monutti (2024), explica que o comando *Order by* pode ser utilizado junto do *select* para ordenar os resultados em ordem crescente ou descrescente. Pode-se observar um exemplo na Figura 5.10:

Figura 5.10 - *Select* com *order by*

```
SELECT nf.numero AS Numero_Nota_Fiscal, nf.num_cpf AS CPF_Cliente,
v.nome AS Nome_Vendedor, nf.data_venda
FROM notas_fiscais nf, vendedores v
WHERE nf.num_matricula = v.matricula
order by nf.data_venda DESC;
```

Resultado da Consulta

Todas as Linhas Extraídas: 8 em 0,081 segundos

	NUMERO_NOTA_FISCAL	CPF_CLIENTE	NOME_VENDEDOR	DATA_VENDA
1	1003	33133133133	Guilherme Reis	07/02/26
2	1006	12312312312	Luana Campos	07/02/26
3	1002	44144144141	Luana Campos	06/02/26
4	1000	12312312312	Bianca Farias	06/02/26
5	1004	55155155155	Alexandre Junior	05/02/26
6	1001	66166166166	Bianca Farias	05/02/26
7	1007	44144144141	Guilherme Reis	04/02/26
8	1005	66166166166	Luana Campos	03/02/26

Fonte: Elaborada pelo autor, 2026

Este *select* é o mesmo da Figura 5.9, o que muda é que os registros foram ordenados pela data da venda. `DESC` ordena do maior para o menor. Os registros foram listados iniciando pela data mais recente até a data mais antiga.

Já o comando *Group By* permite agrupar os resultados com base em uma ou mais colunas. Sendo bem útil quando usado com funções de agregação como *COUNT*, *SUM*, *AVG*, *MIN* e *MAX* (Monutti, 2024). Na Figura 5.11, pode-se visualizar um exemplo:

Figura 5.11 - Select com *group by*

```
SELECT nf.num_matricula, v.nome AS Nome_Vendedor,
count (nf.num_matricula) AS Vendas_Realizadas
from notas_fiscais nf, vendedores v
where nf.num_matricula = v.matricula
group by nf.num_matricula, v.nome
order by Vendas_Realizadas DESC;
```

Resultado da Consulta

SQL | Todas as Linhas Extraídas: 4 em 0,003 segundos

	NUM_MATRICULA	NOME_VENDEDOR	VENDAS_REALIZADAS
1	00003	Luana Campos	3
2	00004	Guilherme Reis	2
3	00002	Bianca Farias	2
4	00001	Alexandre Junior	1

Fonte: Elaborada pelo autor, 2026

Essa consulta exibe a quantidade de vendas realizadas por cada vendedor. Para isso, é preciso agrupar os resultados por meio do comando *Group by*. O trecho "`count (nf.num_matricula) AS Vendas_Realizadas`" é responsável por contar quantas vezes cada número de matrícula aparece nos registros da tabela Notas_Fiscais. O comando *group by*, nesse *select*, agrupou por número de matrícula e nome do vendedor. O comando *order by* serviu para ordenar do maior número de vendas para o menor número de vendas.

5.3.5 Inner Join

INNER JOIN é uma cláusula que permite a junção entre duas ou mais tabelas, desde que haja entrelaçamento entre todas (Devmedia, 2020). Verifica-se um exemplo na Figura 5.12:

Figura 5.12 - Select com inner join

```sql
SELECT c.nome AS Cliente, SUM (inf.quantidade * inf.preco) AS Total_Vendido
from itensnotasfiscais inf
inner join notas_fiscais nf ON
nf.numero = inf.numero
inner join clientes c ON
c.cpf = nf.num_cpf
group by c.nome
order by Total_Vendido DESC;
```

Resultado da Consulta

SQL | Todas as Linhas Extraídas: 5 em 0,009 segundos

	CLIENTE	TOTAL_VENDIDO
1	Marcio Lima	91,73
2	Joana Albuquerque	46,99
3	Luciane Ribeiro	44,99
4	Eric Ricoldi	31
5	Rogério Paizan	20

Fonte: Elaborada pelo autor, 2026

Essa consulta mostra o total vendido por cliente. Isto é, considera o somatório dos valores dos itens comprados de cada nota fiscal. Esse *select* envolveu três tabelas: Clientes, Notas_Fiscais e ItensNotasFiscais.

5.4 CONCLUSÃO

A aprendizagem sobre comandos SQL fundamentais e a realização de consultas bem elaboradas a um banco de dados, dentro de um SGBD do mercado de trabalho (Oracle SQL Developer), proporcionou aos estudantes avanço nos conhecimentos sobre banco de dados relacionais. Tais conhecimentos podem ser aplicados, por exemplo, no desenvolvimento de sistemas *web*, em análises de dados que buscam por informações valiosas e diversas outras aplicações.

6 PROJETO DO 6º SEMESTRE: APLICAÇÃO WEB DE CRUD EM PHP

6.1 CONTEXTUALIZAÇÃO E OBJETIVO

Na disciplina de tópicos especiais em informática, ministrada pelo professor Júlio Fernando Lieira, criou-se uma aplicação *web* em PHP que realiza CRUD de produtos. Como trabalho avaliativo (prova P1) foi proposto que os alunos acrescentassem a esse projeto um CRUD também de fornecedores e a funcionalidade de busca, ou seja, encontrar um produto ou fornecedor desejado por meio de um campo de busca. Houve também a sugestão de implementar paginação das informações na interface da aplicação *web*. O projeto do 6º semestre, portanto, tem como objetivo mostrar e explicar como foi desenvolvido uma aplicação *web* em PHP que realiza CRUD de produtos e fornecedores e possui funcionalidades úteis como campo de busca e paginação.

6.2 SOBRE O PROJETO

Para o desenvolvimento deste projeto o discente utilizou a *Integrated Development Environment* (IDE) Xampp e editor Netbeans. É importante saber que as funcionalidades de CRUD e as funcionalidades de busca e paginação de produtos e fornecedores são implementadas seguindo a mesma lógica. Há poucas diferenças entre os códigos, sendo assim, será apresentado apenas as interfaces e códigos em relação a entidade produtos. Dessa forma, já será possível compreender como a aplicação foi desenvolvida.

6.2.1 Tecnologias utilizadas: XAMPP, PhpMyAdmin e PHP

“O XAMPP nada mais é do que um pacote de *software* livre que facilita a instalação de um servidor *web* local. Ideal para desenvolvedores *web*, ele une Apache, MySQL, PHP e Perl em uma só solução” (Garcia, 2024). Essa ferramenta simplifica a configuração de um ambiente de desenvolvimento *web* local. Isso significa que

através dela os desenvolvedores podem criar, testar e modificar sites e aplicativos antes de publicá-los online.

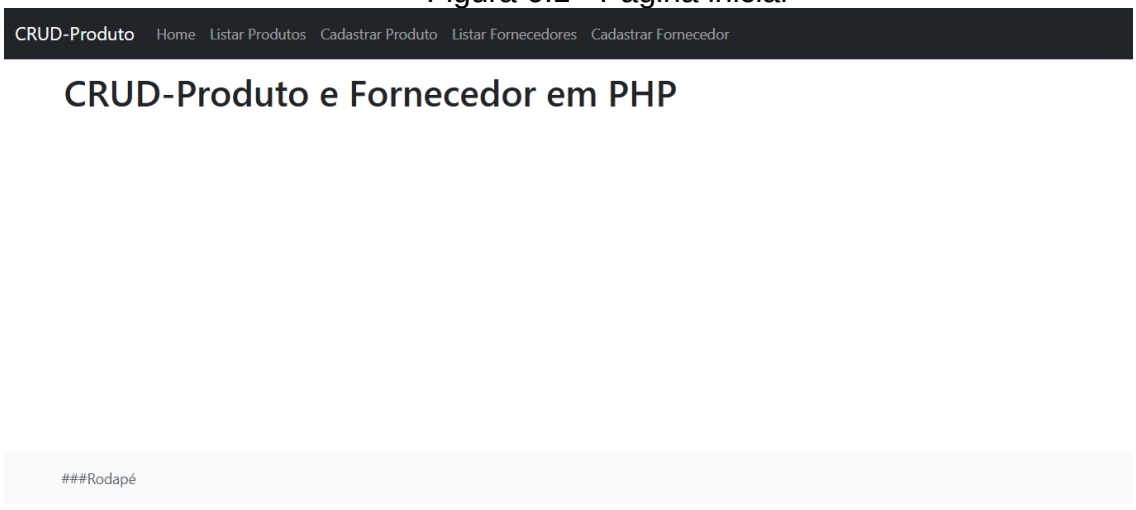
O XAMPP oferece muitas ferramentas úteis como: o Apache que é o servidor HTTP que possibilita a hospedagem de páginas *web*; MySQL ou MariaDB como SGBD; PhpMyAdmin – a interface gráfica para gerenciamento de banco de dados; a linguagem de programação PHP e a linguagem de *script* Perl (Garcia, 2024). O site em português do PHP descreve que o PHP:

"é uma linguagem de *script open source* de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento *web* e que pode ser embutida dentro do HTML" (Php, 2026).

6.2.2 Estrutura da aplicação

Para o *front-end* da aplicação foi utilizado o *framework* Bootstrap em sua versão mais atual até o momento (versão 5.3). E todo o *back-end* foi desenvolvido na linguagem PHP. A pasta do projeto foi criada dentro da pasta *htdocs*. Essa é a pasta do servidor *web* local chamado Apache. As interfaces da aplicação são divididas em três partes principais: o menu de opções, o corpo do site (varia conforme a saída de cada opção da aplicação) e o rodapé. A página inicial, por exemplo, é o arquivo `home.php`. Seu conteúdo é apenas um título HTML. Observa-se a saída do *script* `home.php` na Figura 6.1:

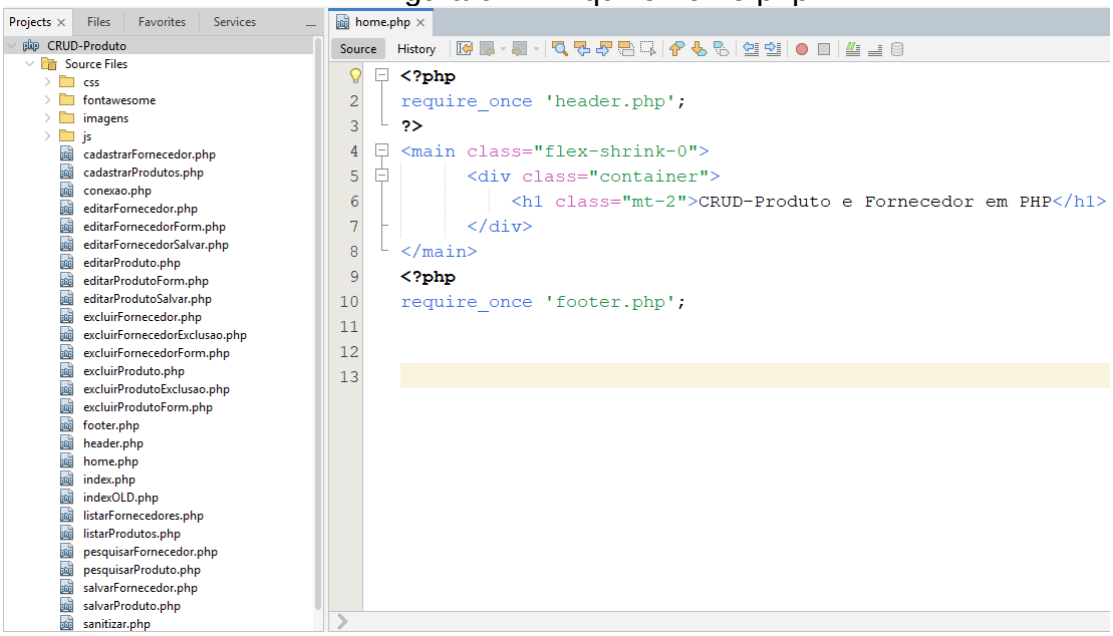
Figura 6.1 - Página inicial



Fonte: Elaborada pelo autor, 2026

A aplicação segue sempre esse modelo: o menu de opções e o rodapé sempre aparecem e o que varia é a parte do meio (conteúdo). Pode-se observar o código do arquivo `home.php` na Figura 6.2:

Figura 6.2 - Arquivo home.php



Fonte: Elaborada pelo autor, 2026

O comando `require_once` do PHP é usado para inserir o conteúdo de outro arquivo PHP. Nesse caso inseriu-se o cabeçalho/menu de opções (arquivo `header.php`) e o rodapé (`footer.php`). Ao lado esquerdo da imagem se encontra todos os arquivos PHP da aplicação e as pastas CSS e JS onde tem os arquivos para utilização do Bootstrap e os arquivos CSS de estilização. Na Figura 6.3 pode-se verificar um trecho do código do arquivo `header.php`:

Figura 6.3 - Links de navegação do arquivo header.php

```
<ul class="navbar-nav me-auto mb-2 mb-md-0">
  <li class="nav-item"> <a class="nav-link" href="home.php">Home</a> </li>
  <li class="nav-item"> <a class="nav-link" href="listarProdutos.php">Listar Produtos</a> </li>
  <li class="nav-item"> <a class="nav-link" href="cadastrarProdutos.php">Cadastrar Produto</a> </li>
  <li class="nav-item"> <a class="nav-link" href="listarFornecedores.php">Listar Fornecedores</a> </li>
  <li class="nav-item"> <a class="nav-link" href="cadastrarFornecedor.php">Cadastrar Fornecedor</a> </li>
</ul>
```

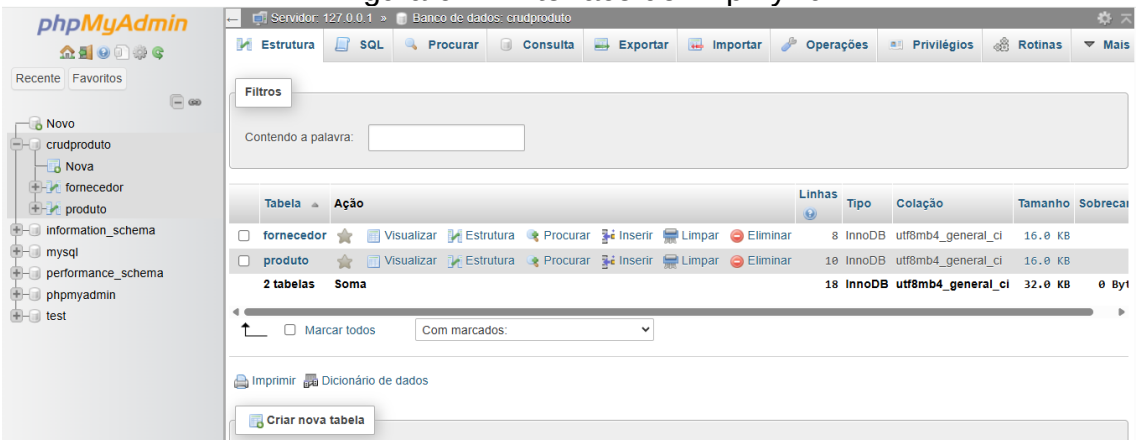
Fonte: Elaborada pelo autor, 2026

Trata-se de uma lista HTML não ordenada que disponibiliza os *links* de navegação para as funções da aplicação. Quando o usuário clica em “Listar Produtos”, por exemplo, será carregado a saída do *script* `listarProdutos.php` na tela.

6.2.3 Banco de dados da aplicação

Para o projeto utilizou-se o banco de dados MySQL e seu gerenciamento foi por meio do PhpMyAdmin. Para se conectar a um banco de dados MySQL, antes, é preciso criar um banco de dados e ter em mãos o nome do banco, nome do usuário e senha. A criação do banco de dados e seu gerenciamento pode ser feito usando o PhpMyAdmin (Gallas, 2026). A Figura 6.4 exibe a interface do PhpMyAdmin:

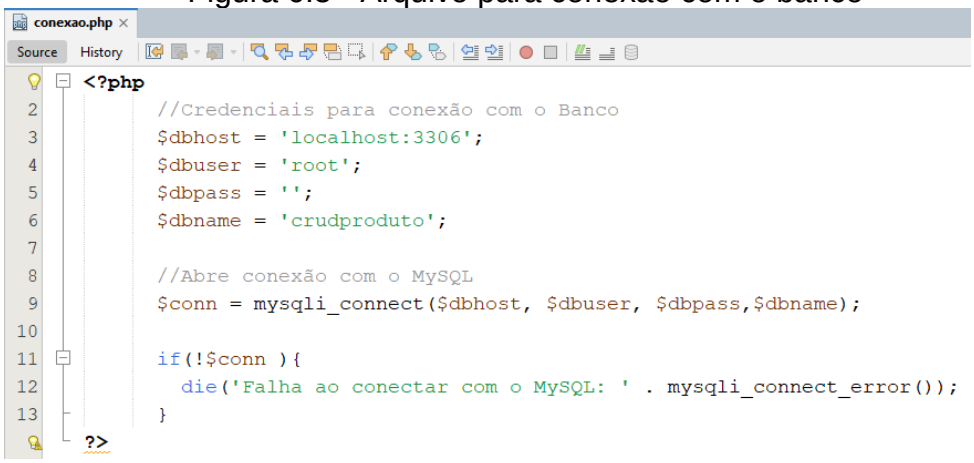
Figura 6.4 - Interface do PhpMyAdmin



Fonte: Elaborada pelo autor, 2026

Nota-se a presença do banco de dados “crudproduto”. Nele há duas tabelas: produto e fornecedor. As funções da aplicação precisam estabelecer uma conexão com o banco de dados para funcionar. Sendo assim, foi criado um arquivo chamado `conexao.php`. Verifica-se seu código na Figura 6.5:

Figura 6.5 - Arquivo para conexão com o banco



Fonte: Elaborada pelo autor, 2026

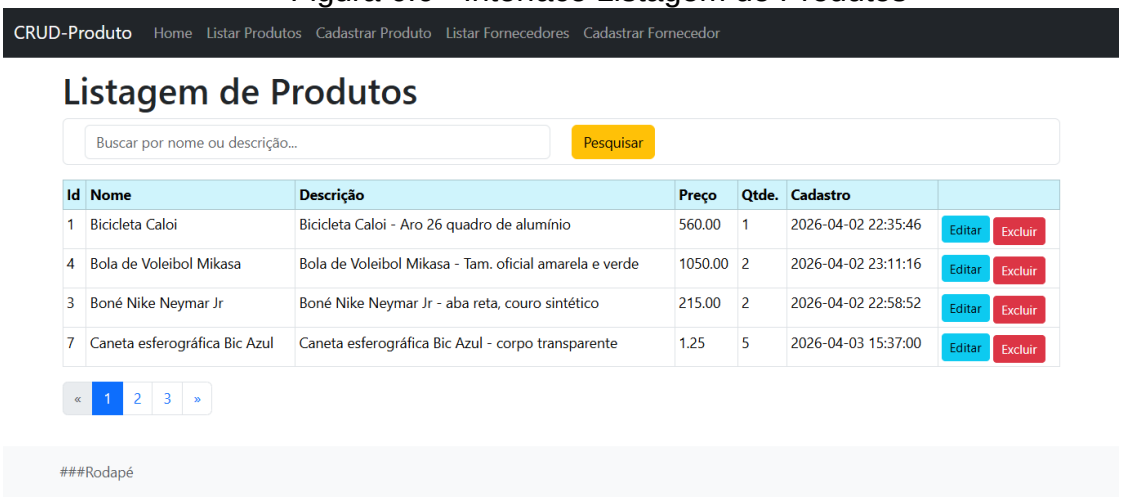
Toda vez que um *script* precisar estabelecer uma conexão com o banco de dados, basta incluir o arquivo `conexao.php` com o comando: `include_once 'conexao.php'`.

6.3 FUNÇÕES DA APLICAÇÃO

6.3.1 Listar produtos

A listagem de produtos é realizada por meio do arquivo `listarProdutos.php`. Os três passos, de forma resumida, para realizar essa operação são respectivamente: realizar conexão com o banco de dados, selecionar/buscar as informações no banco de dados e, por fim, montar a página com as informações que serão exibidas para o usuário. Pode-se conferir, na Figura 6.6, a interface de listagem de produtos da aplicação *web*.

Figura 6.6 - Interface Listagem de Produtos



Fonte: Elaborada pelo autor, 2026

Realizado a conexão com o banco de dados com a inclusão do arquivo `conexao.php`, o próximo passo é buscar as informações no banco de dados e montar a página com os dados, ou seja, exibi-las de forma agradável ao usuário – em forma de tabela. Um trecho de código do arquivo `listarProdutos.php` é exibido na Figura 6.7:

Figura 6.7 - listarProdutos.php: comando select

```php
<?php

include_once 'conexao.php';

$page = isset($_GET['page']) ? (int)$_GET['page'] : 1;

if ($page < 1) {
    $page = 1;
}
$limite = 4;
$offset = ($page - 1) * $limite;

//descobrir o número de páginas
$sql_total = "SELECT COUNT(*) as total FROM Produto";
$result_total = mysqli_query($conn, $sql_total);
$row_total = mysqli_fetch_assoc($result_total);

$total_registros = $row_total['total'];
$total_paginas = ceil($total_registros / $limite);

$sql = "SELECT * FROM Produto ORDER BY nome ASC LIMIT $limite OFFSET $offset";
$result = mysqli_query($conn, $sql); //A query seleciona as linhas da Tabela
```

Fonte: Elaborada pelo autor, 2026

A lógica desse trecho de código consiste em buscar as informações dos produtos no banco de dados respeitando o limite da paginação. Isto é, o comando SQL dentro da variável (`$sql`) faz com que os registros sejam exibidos de 4 em 4 e em ordem alfabética. Agora, basta exibir as informações na tela. Pode-se analisar, na Figura 6.8, o seguinte trecho de código do arquivo `listarProdutos.php`:

Figura 6.8 - Listagem dos produtos

```php
while ($row = mysqli_fetch_assoc($result)){
    echo '<tr>';
    echo '  <td>'.$row['id'].'</td>';
    echo '  <td>'.$row['nome'].'</td>';
    echo '  <td>'.$row['descricao'].'</td>';
    echo '  <td>'.$row['preco'].'</td>';
    echo '  <td>'.$row['quantidade'].'</td>';
    echo '  <td>'.$row['dataCadastro'].'</td>';
    echo '  <td> <a href="editarProduto.php?id='.$row['id'].'" class="btn btn-info btn-sm">Editar</a>
    <a href="excluirProduto.php?id='.$row['id'].'&nome='.$row['nome'].'" '
            . 'class="btn btn-danger btn-sm mt-1">Excluir</a>';
    echo '</tr>';
}
```

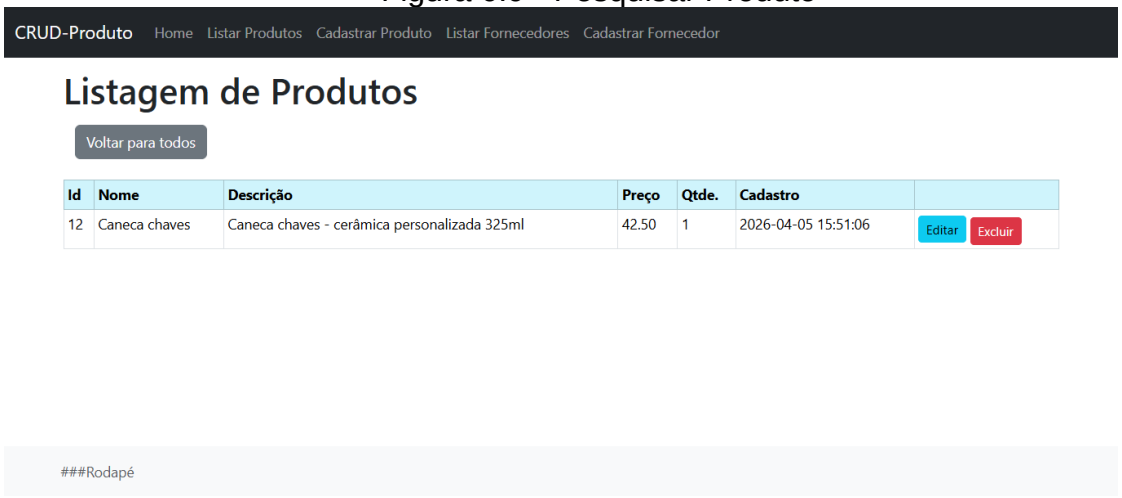
Fonte: Elaborada pelo autor, 2026

Esse trecho de código monta as linhas da tabela com os campos da entidade produto: id, nome, descrição, preço, quantidade e data de cadastro. Também atribui os botões de editar e excluir em cada linha.

A funcionalidade de busca segue a mesma lógica: conexão com o banco, buscar as informações no banco e montar as informações na tela. O que muda é apenas o comando SQL de consulta. Ele exibe apenas os registros que contém, em alguma parte do nome do produto ou da descrição, a *string* que o usuário digitou.

Quando o usuário digita a palavra “chaves”, por exemplo, e clica no botão “Pesquisar” é carregado o conteúdo do arquivo `pesquisarProduto.php` na tela exibindo o(s) produto(s) encontrado(s) com esse termo ou não. Observa-se essa funcionalidade na Figura 6.9:

Figura 6.9 - Pesquisar Produto



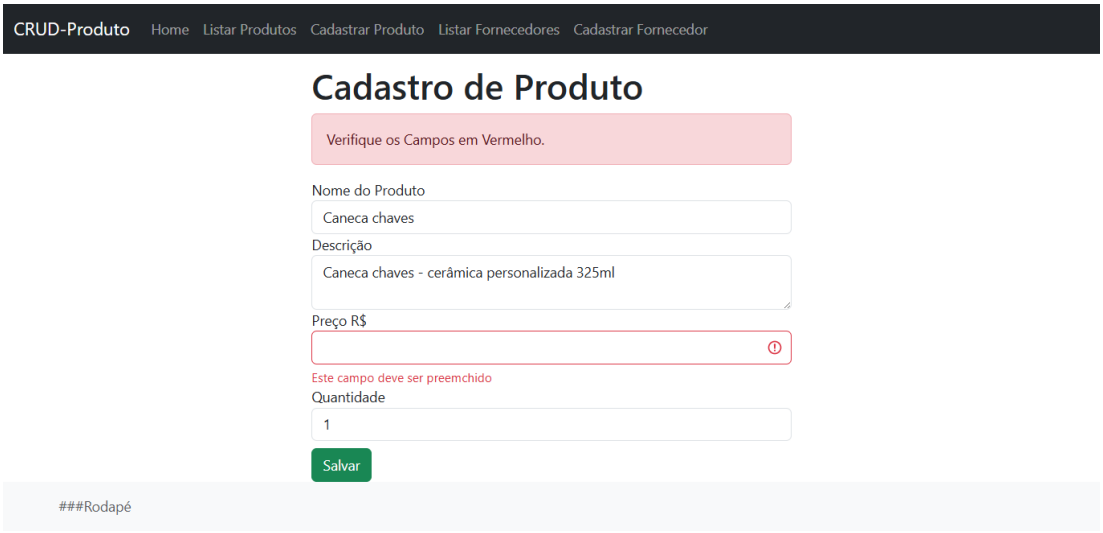
Fonte: Elaborada pelo autor, 2026

No caso de não haver nenhum produto com o termo pesquisado será exibido uma mensagem do tipo: “Nenhum produto encontrado com o termo: chaves”.

6.3.2 Cadastrar produtos

Para o cadastro de novos produtos na aplicação é necessário conter um formulário para a entrada de dados. Quando o usuário submeter as informações é realizado a validação e sanitização desses dados, para então, enviar o comando SQL de inserção no banco. Por fim, exibir uma mensagem de *feedback* ao usuário. Abaixo segue um exemplo dessa funcionalidade na Figura 6.10:

Figura 6.10 - Interface de cadastro de produto



Fonte: Elaborada pelo autor, 2026

O arquivo `cadastrarProdutos.php` contém o código para implementar o *front-end* do formulário de cadastro. Quando o usuário clica em “Salvar” os dados são enviados para o *script* `salvarProduto.php`. Verifica-se a primeira parte do *script* `salvarProduto.php` na Figura 6.11:

Figura 6.11 - salvarProduto.php: parte 1

```php
<?php
  session_start();
  include_once 'sanitizar.php';

  $dadosform = sanitizar($_POST);

  $errovalidacao = false;
  if(empty($dadosform['preco'])){
    $_SESSION['msg'] = '<div class="alert alert-danger" role="alert">'
          . 'Verifique os Campos em Vermelho.</div>';
    $_SESSION['erropreco'] = 'Este campo deve ser preemchido';
    $errovalidacao = true;
  }

  if (isset($_SESSION['form'])){
    unset($_SESSION['form']);
  }

  if ($errovalidacao){
    $_SESSION['form'] = $dadosform;
    header("Location:cadastrarProdutos.php");
    die();
  }
```

Fonte: Elaborada pelo autor, 2026

Essa parte do código inclui a sanitização dos dados vindos do formulário. O arquivo `sanitizar.php` verifica se os dados contêm código malicioso e limpa se houver. Logo em seguida, é feito a validação do campo preço. A estrutura de condição

(`if`), na linha 9, verifica se o campo preço dentro do *array* associativo `$dadosform` está vazio. Caso isso ocorra é preciso “devolver” os dados do formulário para o usuário com um aviso de que o campo preço precisar ser preenchido.

Para devolver os dados ao navegador do usuário é necessário fazer o uso de variáveis de sessão. Isso explica o comando `session_start()` na linha 3. `$_SESSION['msg']` e `$_SESSION['erropreco']` são variáveis de sessão que carregam as mensagens que serão apresentadas no *script* `cadastrarProdutos.php` (o qual exibe o formulário de cadastro).

O `if` da linha 16 limpa os dados da `$_SESSION['form']` que será utilizada para reapresentar os dados ao usuário. O `if` da linha 20 verifica se houve erro de validação. Se sim, a variável de sessão `$_SESSION['form']` recebe os dados já sanitizados da variável `$dadosform`.

Agora, o arquivo `cadastrarProdutos.php` deve estar preparado para receber as mensagens de erro de validação e os dados para reapresentação do formulário que estão armazenados nas variáveis de sessão. Pode-se conferir, na Figura 6.12, o seguinte trecho de código do arquivo `cadastrarProdutos.php`:

Figura 6.12 - cadastrarProdutos.php: preparação para receber mensagens de *feedback*

```php
<?php //Mensagens de Erro ou Sucesso na execução das funções
  if(isset($_SESSION['msg'])){
    echo $_SESSION["msg"];
    unset($_SESSION["msg"]);
  }
?>
```

Fonte: Elaborada pelo autor, 2026

Esse código PHP prepara o arquivo `cadastrarProdutos.php` para exibir mensagens de erro ou sucesso. Na sequência, verifica-se, na Figura 6.13, este outro trecho do código do arquivo `cadastrarProdutos.php`:

Figura 6.13 - cadastrarProdutos.php: preparação para receber os dados do produto

```
<div class="form-group">
  <label for="descricao">Descrição</label>
  <textarea class="form-control" name="descricao">
  <?php if (isset($_SESSION["form"]["descricao"])) echo $_SESSION["form"]["descricao"]; ?></textarea>
</div>
<div class="form-group">
  <label for="preco">Preço</label>
  R$ <input type="text" class="form-control
      <?php if (isset($_SESSION["erropreco"])) echo 'is-invalid'; ?>"
      name="preco" value="<?php if (isset($_SESSION["form"]["preco"])) echo $_SESSION["form"]["preco"]; ?>">
  <div class="invalid-feedback">
      <?php echo $_SESSION["erropreco"]; unset($_SESSION["erropreco"]);?>
  </div>
</div>
```

Fonte: Elaborada pelo autor, 2026

Essa é a forma como se reapresenta as informações que já haviam sido preenchidas. Nesse recorte pode-se observar o campo descrição e o campo de preço. O atributo `value` dos campos de formulário recebem os dados por meio das variáveis de sessão.

Todo esse processo de reapresentar os dados ao usuário ocorre quando há erro de validação. Se não houver erro de validação o arquivo `salvarProduto.php` continua sua execução e envia o comando SQL de inserção do produto ao banco de dados. O *script* da na Figura 6.14 realiza essa etapa:

Figura 6.14 - salvarProduto.php: parte 2

```
30  include_once 'conexao.php';
31
32  $conn -> set_charset("utf8");
33
34  $nome = $dadosform["nome"];
35  $descricao = $dadosform["descricao"];
36  $preco = $dadosform["preco"];
37  $quantidade = $dadosform["quantidade"];
38
39  $sql = "INSERT INTO Produto(nome,descricao,preco,quantidade) "
40       . "VALUES('$nome','$descricao','$preco','$quantidade')";
41  $result = mysqli_query($conn, $sql);
42
43  if(mysqli_affected_rows($conn) != 0){
44    $_SESSION['msg'] = '<div class="alert alert-success" '
45    . 'role="alert">Produto Cadastrado com Sucesso.</div>';
46  }else{
47    $_SESSION['msg'] = '<div class="alert alert-danger" '
48    . 'role="alert">Erro ao cadastrar Produto no Banco!</div>';
49  }
50
51  $conn->close();
52
53  header("Location:cadastrarProdutos.php");
```

Fonte: Elaborada pelo autor, 2026

Note que na linha 39 há o comando SQL que faz o *insert* no banco de dados. Logo em seguida, a página do formulário de cadastro (`cadastrarProdutos.php`) é exibida novamente com uma mensagem de *feedback* ao usuário. Referente a sanitização dos dados, Yara (2025) explica:

> A segurança das aplicações depende diretamente do controle sobre os dados que entram no sistema. Vulnerabilidades como SQL Injection, Command Injection e Cross-Site Scripting (XSS) continuam sendo exploradas por atacantes devido à falta de sanitização adequada dos parâmetros de entrada. Esse descuido pode resultar em vazamento de dados, comprometimento de servidores e execução remota de código malicioso.

“A vulnerabilidade de injeção ocorre quando uma aplicação permite que entradas fornecidas por usuários sejam interpretadas como comandos executáveis dentro do sistema” (YARA, 2025). O SQL *Injection*, por exemplo, trata-se da manipulação de consultas SQL ao banco de dados através de entradas da aplicação. Esses ataques permitem a extração de informações confidenciais, alterações e exclusão de dados críticos (YARA, 2025).

6.3.3 Editar produtos

Quando o usuário clicar no botão editar (na tela de listagem de produtos), será acionado o *script* `editarProduto.php`, o qual receberá o ID do produto a ser editado. Pode-se conferir o código completo do arquivo `editarProduto.php` na Figura 6.15:

Figura 6.15 - Arquivo editarProduto.php

```php
<?php
session_start();
include_once 'sanitizar.php';

$dados = sanitizar($_GET); //Sanitização
$idproduto = $dados['id'];

if (!is_numeric($idproduto)){ //Validação
  die("Id do produto não é numérico!");
}

include_once 'conexao.php';

$sql = "SELECT * FROM Produto WHERE id={$idproduto}";
$result = mysqli_query($conn, $sql); //A query seleci

if(mysqli_affected_rows($conn) != 1){
  die('Falha ao recuperar dados do produto');
}

$produto = mysqli_fetch_assoc($result);

$_SESSION['form'] = $produto;
header("Location:editarProdutoForm.php");

$conn->close(); //Fecha a conexão com o Banco
```

Fonte: Elaborada pelo autor, 2026

Esse arquivo realiza, em primeiro lugar, a sanitização dos dados e validação do ID do produto. Passado essa etapa, abre-se uma conexão com o banco de dados e é efetuado um comando *SELECT* que busca o produto pelo ID. Os dados desse produto são armazenados, por fim, na variável de sessão `$_SESSION['form']`. Ela permite que os dados do produto sejam exibidos no formulário de edição (`editarProdutoForm.php`). Verifica-se a interface de editar Produtos na Figura 6.16:

Figura 6.16 - Interface editar produto

Fonte: Elaborada pelo autor, 2026

O arquivo `editarProdutoForm.php` implementa a mesma lógica do arquivo `cadastrarProdutos.php` (pode-se analisar a Figura 6.10, Figura 6.12 e Figura 6.13), no qual, implementa os campos de formulário na tela e recebe os dados do produto e as mensagens de *feedback* via variáveis de sessão. Quando o usuário realiza as modificações e clica em “Salvar” os dados serão enviados para o *script* `editarProdutoSalvar.php`. Esse arquivo, em primeiro lugar, realiza a sanitização dos dados e validação do campo preço (verifica se está vazio ou não) e se os dados advindos do formulário de edição não contêm alterações. Havendo erro de campo preço não preenchido e/ou os dados do produto não foram editados, será reapresentado a página do formulário de edição com os dados do produto e as mensagens de alerta. Verifica-se um trecho do código do arquivo `editarProdutoSalvar.php`, que realiza esse processo, na Figura 6.17:

Figura 6.17 - editarProdutoSalvar.php: parte 1

```
13      unset($_SESSION['form']['dataCadastro']);
14
15      if (empty(array_diff_assoc($dadosform,$_SESSION['form']))){
16          $_SESSION['msg'] = '<div class="alert alert-warning" role="alert">'
17                      . 'Você NÃO alterou os Dados. Verifique!</div>';
18          $errovalidacao = true;
19      }
20
21      if (isset($_SESSION['form'])){
22        unset($_SESSION['form']);
23      }
24
25      if ($errovalidacao){  //Houve erro na validacao
26        $_SESSION['form'] = $dadosform;
27        header("Location:editarProdutoForm.php"); //Retorna ao Formulário
28        die();
29      }
```

Fonte: Elaborada pelo autor, 2026

O `if` da linha 25 verifica se houve erro de validação. Se sim, o usuário é redirecionado a página do formulário de edição com reapresentação dos dados e mensagens de *feedback*. Não havendo erros de validação, basta abrir uma conexão com o banco de dados e enviar o comando SQL de *update* dos dados. Pode-se observar esse processo na Figura 6.18:

Figura 6.18 - editarProdutoSalvar.php: parte 2

```php
include_once 'conexao.php';

$conn -> set_charset("utf8");

$sql = "UPDATE Produto SET nome='".$dadosform["nome"]."',"
      . "descricao='".$dadosform["descricao"]."',preco='".$dadosform["preco"]."',"
      . "quantidade='".$dadosform["quantidade"]."' WHERE id='".$dadosform["id"]."'";

$result = mysqli_query($conn, $sql); //A query seleciona as linhas da Tabela

if(mysqli_affected_rows($conn) != 0){
  $_SESSION['msg'] = '<div class="alert alert-success" role="alert">'
  . 'Produto Atualizado com Sucesso.</div>';
}else{
  $_SESSION['msg'] = '<div class="alert alert-danger" role="alert">'
  . 'Erro ao Atualizar Produto no Banco!</div>';
}

$conn->close(); //Fecha a conexão com o Banco

header("Location:listarProdutos.php");
```

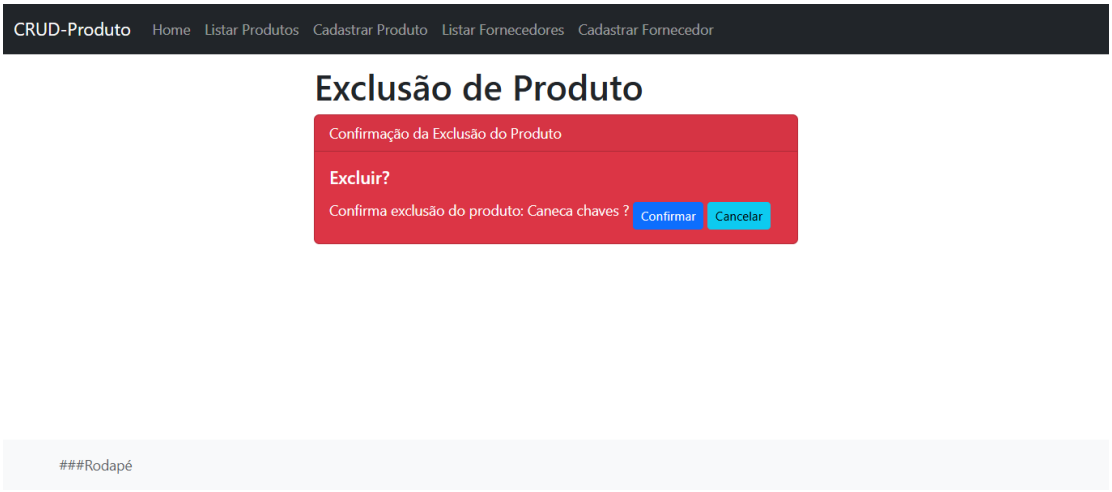
Fonte: Elaborada pelo autor, 2026

Após a atualização dos dados será carregado a página de listagem de produtos com uma mensagem de sucesso.

6.3.4 Excluir produtos

O fluxo para a exclusão de um produto ocorre da seguinte forma: quando o usuário clica no botão excluir (na tela de listagem de produtos) são enviados o ID e o nome do produto para o *script* `excluirProduto.php` que realiza a sanitização dos dados e validação do campo ID do produto, e logo em seguida, envia esses dados para o arquivo `excluirProdutoForm.php`. Esse script exibe um *card* para pedir a confirmação da exclusão do produto ao usuário. Observa-se, essa funcionalidade, na Figura 6.19:

Figura 6.19 - Exclusão de produto



Fonte: Elaborada pelo autor, 2026

Se confirmada a exclusão: o arquivo `excluirProdutoExclusao.php` é acionado e realiza mais uma vez a sanitização e validação do campo ID e envia o comando SQL de exclusão para o banco. Por fim, retorna a tela de listagem de produtos exibindo a mensagem: “Produto Excluído com sucesso”.

6.4 CONCLUSÃO

O desenvolvimento desta aplicação *web*, utilizando PHP em conjunto com o *framework* Bootstrap para a construção das interfaces, proporcionou aos alunos a aquisição de conhecimentos relevantes na área de desenvolvimento *web*. Ao final do projeto, os estudantes tornam-se aptos a desenvolver sistemas para diversas finalidades, uma vez que a análise e prática com os códigos da aplicação contribuíram para a construção de uma base sólida de aprendizado. Além disso, o projeto serve como incentivo para o contínuo aprimoramento das habilidades e das boas práticas no desenvolvimento de sistemas.

7 CONCLUSÃO

O portfólio acadêmico configura-se como um modelo de Trabalho de Graduação (TG) aceito pela instituição Fatec Lins – Prof. Antonio Seabra, caracterizando-se como uma abordagem que possibilita a consolidação, em um único documento, dos projetos mais relevantes desenvolvidos ao longo de cada semestre letivo do curso de ADS. Nesse contexto, o presente trabalho teve como principal objetivo evidenciar, de forma estruturada e reflexiva, os conhecimentos adquiridos durante toda a trajetória acadêmica. Como complemento a esta monografia, foi desenvolvido também um portfólio online, o qual apresenta, de maneira mais sucinta e dinâmica, os projetos descritos, ampliando o alcance e a acessibilidade das produções realizadas.

A organização e a contextualização da descrição de cada projeto possibilitaram ao discente não apenas registrar suas produções, mas também reconhecer sua evolução ao longo do curso, tanto em termos conceituais quanto técnicos. Esse processo evidencia a importância da articulação entre teoria e prática, materializada por meio de projetos disciplinares, interdisciplinares e demais atividades acadêmicas, que contribuem significativamente para a formação de um profissional qualificado e competitivo no mercado de trabalho. Dessa forma, destaca-se que o estudante não se limitou à escrita de códigos, mas desenvolveu, de maneira consistente, sua capacidade de análise, resolução de problemas e pensamento crítico.

Os tópicos abordados em cada capítulo foram estruturados com o propósito de promover uma análise sob as perspectivas técnica e pedagógica. Assim, buscou-se apresentar não apenas o processo de desenvolvimento dos projetos, mas também as tecnologias empregadas, os conceitos teóricos envolvidos, os objetivos propostos, bem como os conhecimentos adquiridos e a utilidade prática de cada solução desenvolvida. Essa abordagem metodológica permite que futuros alunos e pesquisadores compreendam, de forma clara, o percurso formativo proporcionado pelo curso de ADS, servindo como referência e fonte de consulta.

Diante do exposto, conclui-se que o portfólio acadêmico cumpre o papel de reunir, organizar e compartilhar os conhecimentos adquiridos pelo discente ao longo da graduação, evidenciando sua formação técnica e intelectual. Além disso, constitui-se como uma importante ferramenta de apresentação profissional ao mercado de trabalho na área de ADS, ao mesmo tempo em que busca contribuir para a divulgação

do curso junto à comunidade e para a orientação de futuros ingressantes, demonstrando, de forma concreta, as competências e habilidades desenvolvidas durante a formação acadêmica.

REFERÊNCIAS

AMARAL, M. C. P. F. **Requisitos de UX e UI em sistemas web**. 2024. Trabalho de Conclusão de Curso (curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - Unidade Acadêmica Especializada em Ciências Agrárias, Universidade Federal do Rio Grande do Norte, Macaíba, 2024. Disponível em: https://repositorio.ufrn.br/items/6a17f11b-1722-4ab3-bbe4-5cb4d1c43e31. Acesso em: 27 de out. de 2025.

AMOASEI, J. **Estruturas de dados: uma introdução**. 2022. Disponível em: https://www.alura.com.br/artigos/estruturas-de-dados-introducao?utm_term=&utm_campaign=topo-aon-search-gg-dsa-artigos_conteudos&utm_source=google&utm_medium=cpc&campaign_id=11384329873_185316875709_774786248563&utm_id=11384329873_185316875709_774786248563&hsa_acc=7964138385&hsa_cam=topo-aon-search-gg-dsa-artigos_conteudos&hsa_grp=185316875709&hsa_ad=774786248563&hsa_src=g&hsa_tgt=aud-527303763294:dsa-2439320411982&hsa_kw=&hsa_mt=&hsa_net=google&hsa_ver=3&gad_source=1&gad_campaignid=11384329873&gbraid=0AAAADpqZICNqxy1NpYpMg1uT9SQSrAhr&gclid=CjwKCAiA8vXIBhAtEiwAf3B-g_SY6vleUaTocr3rb9WdVImiWCGuFUs9G09ThIsLqeOo8BKFFiSEJhoCw20QAvD_BwE. Acesso em: 19 nov. 2025.

AUGUSTO, C. **Web Services: O que é, como funciona e os protocolos SOAP e REST**. 2019. Disponível em: http://ninjadolinux.com.br/web-services-soap-e-rest/#disqus_thread. Acesso em: 05. nov. 2025.

ALVES, L. **Um breve resumo do Spring e Spring Boot**. 2024. Disponível em: https://www.dio.me/articles/um-breve-resumo-do-spring-e-spring-boot. Acesso em: 05. nov. 2025.

ANJOS, G. **Tipos de linguagem do SQL e seus comandos**. 2023. Disponível em: https://www.dio.me/articles/tipos-de-linguagem-sql. Acesso em: 23 mar. 2026.

AZEVEDO, P. M.; GIBERTONI, D. A importância do design centrado no usuário em metodologias ageis como requisito de usabilidade. **Revista Interface Tecnológica**, Taquaritinga, SP, v. 17, n. 2, p. 293–305, 2020. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/986. Acesso em: 15 fev. 2026.

BOOTSTRAP. Getbootstrap, 2025. Documentação. Disponível em: https://getbootstrap.com/. Acesso em: 09 de out. de 2025.

BALIEIRO, R. **Estrutura de dados**. Rio de Janeiro: SESES, 2015. Disponível em: https://hood.com.br/new/filegator/repository/Estrutura%20de%20Dados/10%20Livro%20Estrutura%20de%20Dados.pdf. Acesso em: 10 de nov. de 2025.

DEVMEDIA. **SQL: Trabalhando com INNER JOIN**. 2020. Disponível em: https://www.devmedia.com.br/sql-inner-join/41230. Acesso em: 29 mar. 2026.

FABRO, C. **O que é APIs e para que serve? Cinco perguntas e respostas.** 2020. Disponível em: https://www.techtudo.com.br/listas/2020/06/o-que-e-api-e-para-que-serve-cinco-perguntas-e-respostas.ghtml. Acesso em: 04 nov. 2025.

GALLAS, G. **PHP e MySQL: Como conectar e consultar o MySQL com PHP**. 2026. Disponível em: https://www.homehost.com.br/blog/tutoriais/php/como-acessar-e-consultar-um-banco-mysql-usando-php/. Acesso em: 06 abr. 2026.

GARCIA, G**. XAMPP: O que é, como funciona, vantagens e instalação da ferramenta**. 2024.Disponível em: https://mercadoonlinedigital.com/blog/xampp/. Acesso em: 05 abr. 2026.

GOTARDO, R. A. **Linguagem de programação I**. Rio de Janeiro: SESES, 2015. Disponível em: https://www.academia.edu/25554061/LIVRO_PROPRIETARIO_Linguagem_de_Programa. Acesso em: 20 de abr. de 2026.

GUEDES, P. P. C. et al. Frameworks JavaScript: análise comparativa dos principais construtores de interfaces web modernas. **Revista de Tecnologia da Informação da Faculdade Lourenço Filho**, Fortaleza – CE, v. 2, n. 1, p. 1–22, jan./jun. 2021. Disponível em: https://multiversa.edu.br/docs/revista-cientifica/ARTIGO_4_FRAMEWORKS%20JAVASCRIPT_AN%C3%81LISE%20COMPARATIVA_PEDRO_GUESDE.pdf. Acesso em: 27 out. 2025.

JUNIOR, E. A. G.; ROCHA, R. D.; MACIEL, R. S. Desenvolvimento de API Rest com Spring Boot. **Revista Científica do Centro Universitário do Rio São Francisco**. v. 15. n. 29. p. 499-525. 2021. Disponível em: https://www.publicacoes.unirios.edu.br/index.php/revistarios/article/view/102. Acesso em: 05 nov. 2025.

LACERDA, W. S. **Introdução à Linguagem C**. 2002. Curso de Pós-Graduação “Lato Sensu” (Especialização) a Distância: Administração em Redes Linux – Universidade Federal de Lavras (UFLA), Lavras, 2002. Disponível em: https://www.academia.edu/15015935/CURSO_DE_P%C3%93S_GRADUA%C3%87%C3%83O_LATO_SENSU_ESPECIALIZA%C3%87%C3%83O_A_DIST%C3%82NCIA_ADMINISTRA%C3%87%C3%83O_EM_REDES_LINUX_INTRODU%C3%87%C3%83O_%C3%80_LINGUAGEM_C. Acesso em: 10 de nov. de 2025.

LUCCHESI, C. **Dominando a análise e desenvolvimento de sistemas**: Guia Completo para Iniciantes. São Paulo: Claudio Lucchesi, 2025. 95 p. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=abc7EQAAQBAJ&oi=fnd&pg=PA6&dq=analista+desenvolvedor+de+sistemas&ots=jhwbmmnJKa&sig=Ghu62emkNDJmbvOGo1u0U7t6qvo&redir_esc=y#v=onepage&q&f=false. Acesso em: 01 de nov. de 2025.

LETT, J. **Bootstrap 4 Grid Flexbox Tutorial with Layout Examples**. 2024. Disponível em: https://bootstrapclasses.com/bootstrap-4-grid-explained-examples/. Acesso em: 18 de out. de 2025.

MARIANO, D. **Bootstrap 5 – Guia Rápido para Iniciantes.** Lagoa Santa: Alfahelix Publicações, 2022a. Disponível em: https://books.google.com.br/books?id=1idnEAAAQBAJ&lpg=PA1&hl=pt-BR&pg=PA1#v=onepage&q&f=false. Acesso em: 06 de out. de 2025.

- . **Layout no Bootstrap 5**: Capítulo 2. 2022b. Disponível em: https://diegomariano.com/layout-no-bootstrap-5/. Acesso em: 11 out. 2025.

MACHADO, H. G. **SQL*:* aprenda a utilizar a chave primária e a chave estrangeira**. 2020. Disponível em: https://www.devmedia.com.br/sql-aprenda-a-utilizar-a-chave-primaria-e-a-chave-estrangeira/37636. Acesso em: 23 mar. 2026.

MAK, G. **Spring mvc framework**. In: Spring Recipes: A Problem-Solution Approach. [S.l.]: Springer, 2008. p. 321–393. Disponível em: https://link.springer.com/chapter/10.1007/978-1-4302-0623-1_10. Acesso em: 04 nov. 2025.

MONUTTI, D. **Select no SQL: o comando mais importante do SQL**. 2024. Disponível em: https://www.hashtagtreinamentos.com/select-no-sql#resumo-comando-select-no-sql. Acesso em: 28 mar. 2026.

MIRANDA, L**. Crud: o que é e como funciona na programação**. 2024. Disponível em: https://querobolsa.com.br/revista/crud-o-que-e. Acesso em: 22 nov. 2025.

MORAIS, L. **Qualidade de Software – Engenharia de Software 29**. 2010. Disponível em: https://www.devmedia.com.br/qualidade-de-software-engenharia-de-software-29/18209. Acesso em 30 out. 2025.

MONTEIRO, Enio de Jesus Pontes et al. Recomendações para adoção de padrões de usabilidade e responsividade no desenvolvimento de aplicações web. ***Brazilian Journal of Development***, Curitiba, v. 6, n. 5, p. 24790-24819, maio 2020. Disponível em: https://ojs.brazilianjournals.com.br/ojs/index.php/BRJD/article/view/9600. Acesso em: 25 jun. 2026.

OLIVEIRA, D.; DIAS, G. **Saiba tudo sobre SQL: a linguagem padrão para trabalhar com banco de dados relacionais**. 2019. Disponível em: https://www.alura.com.br/artigos/o-que-e-sql. Acesso em: 16 fev. 2026.

PHP. **O que é PHP e o que ele pode fazer?**. 2026. Disponível em: https://www.php.net/manual/pt_BR/introduction.php. Acesso em: 05 abr. 2026.

ROCKETSEAT. **Desvendando o Hibernate: torne o mapeamento objeto-relacional fácil em Java**. 2025. Disponível em: https://www.rocketseat.com.br/blog/artigos/post/introducao-hibernate-simplifique-orm-java. Acesso em: 08. nov. 2025.

SILVA, A. A. P. **Design responsivo: técnicas, frameworks e ferramentas.** 2014. Projeto de Graduação (Bacharelado em Sistemas de Informação) – Escola de Informática Aplicada, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2014. Disponível em: https://bsi.uniriotec.br/wp-content/uploads/sites/31/2020/05/201412Almeida.pdf. Acesso em: 16 out. 2025.

SILVA, M. S. **Fundamentos de HTML5 e CSS3.** São Paulo: Novatec, 2018. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=lang_pt&id=ZyJyDwAAQBAJ&oi=fnd&pg=PA1&dq=Conceito+de+HTML,+CSS&ots=L3zjLkPXdm&sig=0CPd1tbA8n7bvn1ZTWgRMoeFCuU&redir_esc=y#v=onepage&q&f=false. Acesso em: 27 out. 2025.

- . **CSS3: Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3.** São Paulo: Novatec, 2019. Disponível em: https://books.google.com.br/books?id=dMy-DwAAQBAJ&hl=pt-BR&source=gbs_navlinks_s. Acesso em: 27 out. 2025.

- . **Web Design Responsivo: aprenda a criar sites que se adaptam automaticamente a qualquer dispositivo, desde desktops até telefones celulares.** São Paulo: Novatec, 2014. Disponível em: https://books.google.com.br/books?id=HNHHAwAAQBAJ&lpg=PA9&ots=KQBxRIAM3I&dq=Silva%2C%20M.S.%20Web%20Design%20Responsivo&lr&hl=pt-BR&pg=PA4#v=onepage&q&f=false. Acesso em: 26 abr. 2026.

SILVA, Z. M. **Desenvolvimento de API REST de autenticação utilizando Spring para arquiteturas de microsserviços.** 2023. Trabalho de Conclusão de Curso (Bacharelado em Engenharia da Computação) – Universidade Federal da Paraíba, João Pessoa, 2023. Disponível em: https://repositorio.ufpb.br/jspui/bitstream/123456789/31769/1/Zaqueu%20Moura%20da%20Silva_TCC.pdf. Acesso em: 04 nov. 2025.

SALAH, D.; PAIGE, R.; CAIRNS, P. Integrating agile development processes and user centred design: a place for usability maturity models? In: SAUER, S. et al. (eds.). HCSE 2014. Cham: Springer, 2014. p. 108–125. (Lecture Notes in Computer Science, v. 8742).

SCHILDT, H**. C completo e total**. 3. ed. rev. e atual. Tradução e revisão técnica de Roberto Carlos Mayer. São Paulo: Makron Books, 1996. Disponível em: https://d1wqtxts1xzle7.cloudfront.net/59132884/c_completo_e_total_3%C2%AA_ed._-_herbert_schildt_-_makron_books20190505-82148-1q58yep.pdf?1738383550=&response-content-disposition=inline%3B+filename%3DC_completo_e_total_3a_ed_herbert_schildt.pdf&Expires=1777236308&Signature=BTyR7e83JcCm8MHvdCfhmFOAhiDhoE4hsG-hILQI60qgAwu7m1w3b0VXWYQqVra9JJBGsOWD-L9wZGkM2FBUv4FjsB~19n9v1HDg59YxLp-BsN7N~WnC7mtcMgEtUXvwIeM0Xgrnmov-TvXC89BfGfplqvZarobaNvnR3mTB00y~K5tVyXfhZ3k4SiNMLpMYpMd2~KXLJ-F9y-ynbtkHaVYwSHJ36ymlyoMOA4EkMunFcGj40rmgWnxnM1KDkDX~2wLiqyGgjH6e9ozciY8fvNSEShBs8-

hqMAD7hgmdfMvZ3wFvpIphw4e1KjmJN~zBERSdew4C3kQv2LUnED~P2Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. Acesso em: 11 de out. de 2025.

SORAGE, R. **Além do Código: A Importância Vital das Estruturas de Dados**. 2023. Disponível em: https://medium.com/@rsorage/al%C3%A9m-do-c%C3%B3digo-a-import%C3%A2ncia-vital-das-estruturas-de-dados-bb20d54389d0. Acesso em: 19 nov. 2025.

TEORIA das cores: o guia definitivo para designers. UXDI – ESPM + UX Design Institute. 2025. Disponível em: https://uxdi.espm.br/teoria-das-cores-o-guia-definitivo-para-designers/. Acesso em: 15 fev. 2026.

YARA, Paula. A importância da sanitização de parâmetros de entrada para evitar vulnerabilidades de injeção. **IVY NEWS,** 2025. Disponível em: https://news.ivy.com.br/index.php/2025/03/12/a-importancia-da-sanitizacao-de-parametros-de-entrada-para-evitar-vulnerabilidades-de-injecao/. Acesso em: 26 jun. 2026.

ZENARO, R. D. S. **Análise e desenvolvimento de sistemas**. São Paulo: Senac, 2025. 84 p. Disponível em: https://books.google.com.br/books?id=5p-REQAAQBAJ&newbks=0&dq=an%C3%A1lise+e+desenvolvimento+de+sistemas&hl=pt-BR&source=gbs_navlinks_s. Acesso em: 01 nov. 2025.