CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE LINS PROF. ANTÔNIO SEABRA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

CAMILA GIOVANA TELECIO DOS SANTOS

PORTFÓLIO ACADÊMICO

LINS/SP
2º SEMESTRE/2025

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE LINS PROF. ANTÔNIO SEABRA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

CAMILA GIOVANA TELECIO DOS SANTOS

PORTFÓLIO ACADÊMICO

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Lins Prof. Antônio Seabra, para obtenção do Título de Tecnólogos em Sistemas Para Internet

Orientador: Prof. Dr. Thiago Seti Patrício.

LINS/SP
2º SEMESTRE/2025

Santos, Camila Giovana Telecio dos

S237p Portfólio Acadêmico / Camila Giovana Telecio dos Santos. — Lins, 2025.

74f.

Trabalho de Conclusão de Curso (Tecnologia em Sistemas para Internet) — Faculdade de Tecnologia de Lins Professor Antonio Seabra: Lins, 2025.

Orientador(a): Dr. Thiago Seti Patrício

1. Portfólio Acadêmico. 2. Potfólio Digital. 3. Projetos Acadêmicos. 4. Tecnologia da Informação. 5. Tecnologias Web. I. Patrício, Thiago Seti . II. Faculdade de Tecnologia de Lins Professor Antonio Seabra. III. Título.

CDD 004.61

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

CAMILA GIOVANA TELECIO DOS SANTOS

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 Sistemas Para Internet sob orientação do Prof. Dr. Thiago Seti Patrício.

Data de aprovação: 10/12/2025

_____

Prof. Dr. Thiago Seti Patrício

_____

Prof. Dr. Alciano Gustavo G. Oliveira

_____

Profª. Drª. Alyssa Carolina Barbosa M. Gedo

AGRADECIMENTOS

Chegar até a conclusão deste Trabalho de Graduação foi uma jornada que transcendeu o âmbito acadêmico, tornando-se um testemunho do apoio, amor e incentivo que recebi de muitas pessoas.

À minha mãe, Roseli Telecio, pelo suporte incondicional e por ser meu alicerce em todos os momentos.

À minha família, em especial minha avó, Maria Telecio, e em memória do meu avô, Arlindo Telecio, que partiu enquanto eu me dedicava a reta final deste TG, deixando uma saudade que se une com a força que ele sempre me deu.

Aos meus amigos de curso, que se tornaram companheiros de lutas e vitórias, meu sincero obrigada, pelas risadas e por toda a cumplicidade que tornou essa caminhada mais leve e feliz.

Um agradecimento especial ao meu cachorro, Theo, cuja lealdade silenciosa e alegria contagiante foram um antídoto contra o estresse e uma fonte inesgotável de conforto durante as longas horas de trabalho.

Por fim, deixo aqui minha gratidão a Fatec de Lins e todos os profissionais brilhantes que compartilharam do seu conhecimento durante todos esses anos.

A todos que, de alguma forma, estiveram ao meu lado, o meu mais profundo e sincero, obrigada.

Camila Giovana Telecio dos Santos

RESUMO

Como forma de consolidar a trajetória acadêmica, este estudo documenta a evolução profissional durante a graduação em Sistemas para Internet. O objetivo principal foi consolidar, em um único espaço digital, a progressão das competências técnicas desenvolvidas ao longo do curso, transformando projetos acadêmicos em evidências tangíveis de aprendizagem e constituindo um material essencial para a formação e crescimento pessoal. A metodologia adotada consistiu na organização cronológica por semestres, selecionando projetos representativos de disciplinas fundamentais que demonstram o crescimento contínuo do aprendizado. Partindo de Padrões de Projeto de Sítios Internet I e II, avançou-se para Programação de Sítios Internet III, seguido por Banco de Dados e Internet II, posteriormente Tópicos Especiais em Sistemas para Internet II, e culminando em Desenvolvimento para Dispositivos Móveis. Os resultados revelam não apenas o domínio progressivo de tecnologias web, mas principalmente a capacidade de aplicação integrada dos conhecimentos na resolução de problemas práticos. Conclui-se que o portfólio transcende sua função de vitrine profissional para tornar-se um instrumento que unifica toda a experiência conquistada durante a formação acadêmica.

Palavras-chave: Portfólio Acadêmico. Portfólio Digital. Projetos Acadêmicos. Tecnologia Da Informação. Tecnologias Web.

ABSTRACT

As a means of consolidating the academic trajectory, this study documents the professional evolution during graduation in Internet Systems. The main objective was to consolidate, in a single digital space, the progression of technical skills developed throughout the course, transforming academic projects into tangible evidence of learning and constituting essential material for personal and educational growth. The methodology adopted consisted of chronological organization by semesters, selecting representative projects from fundamental disciplines that demonstrate continuous learning growth. Starting with Website Design Patterns I and II, it advanced to Website Programming, followed by Database and Internet, subsequently Special Topics in Internet Systems, and culminating in Mobile Development. The results reveal not only the progressive mastery of web technologies but primarily the capacity for integrated application of knowledge in solving practical problems. In conclusion, the portfolio transcends its function as a professional showcase to become an instrument that unifies all experience achieved during academic education.

Keywords: Academic Portfolio. Digital Portfolio. Academic Projects. Information Technology. Web Technology.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API -	*Application Programming Interface*
CRUD -	*Creat Read Update Delete*
CSS -	*Cascading Style Sheets*
DDL -	*Data Definition Language*
DML -	*Data Manipulation Language*
DOM -	*Document Object Model*
IDE -	*Integrated Development Environment*
HTML -	*Hypertext Markup Language*
MER -	Modelo de Entidade e Relacionamento
MySQL -	*My Structured Query Language*
SGBD -	Sistema de Gerenciamento de Banco de Dados
SPA -	*Single Page Applications*
SQL -	*Structured Query Language*
TG -	Trabalho de Graduação
WCAG -	*Web Content Accessibility Guidelines*
XML -	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O portfólio acadêmico digital configura-se como uma ferramenta estratégica para estudantes e profissionais da área de tecnologia, por possibilitar o registro e a apresentação sistematizada dos projetos desenvolvidos ao longo da formação acadêmica. Para além de uma mera vitrine de trabalhos, consolida-se como um instrumento de aprendizagem contínua, capaz de evidenciar a evolução técnica do discente, suas escolhas criativas e a aplicação de ferramentas e estratégias na resolução de problemas. De acordo com Machado, e Sakalauskas (2019), o portfólio digital amplia a autonomia do estudante e promove uma avaliação mais reflexiva, favorecendo a consolidação das competências adquiridas.

Ao longo da graduação em Sistemas para Internet na Faculdade de Tecnologia de Lins - Prof. Antônio Seabra (FATEC Lins), diversos projetos práticos foram desenvolvidos, abrangendo desde os fundamentos de criação de sítios web até a construção de aplicações interativas com persistência de dados. Diante desse contexto, evidenciou-se a necessidade de consolidar tais conhecimentos em um único espaço digital que, para além de compilar os trabalhos desenvolvidos semestre a semestre, demonstrasse claramente a progressão das habilidades adquiridas ao longo do curso.

Este Trabalho de Graduação (TG) tem como objetivo principal apresentar a construção do portfólio acadêmico digital da autora, estruturado por semestres e destacando um projeto técnico relevante de cada etapa da formação. Para isso, cada capítulo foi elaborado com base em uma disciplina prática cursada no semestre correspondente, de modo a documentar o processo de aprendizado, as soluções implementadas e a aplicação dos conhecimentos adquiridos.

Em um mercado de trabalho cada vez mais competitivo e visual, um portfólio bem construído pode representar uma vantagem significativa em processos seletivos, além de constituir um recurso valioso para *networking* profissional e acadêmico.

No segundo capítulo, a disciplina de Padrões de Projeto de Sítios Internet I introduziu os fundamentos da construção de páginas web por meio do uso da linguagem de marcação *Hypertext Markup Language* (HTML5) e das folhas de estilo *Cascading Style Sheets* (CSS3). O projeto desenvolvido denominado “Mirror Fashion”, consistiu em um *E-commerce* estático, com foco na organização de layout, semântica do código e aplicação dos conceitos básicos de acessibilidade e usabilidade.

Já no terceiro capítulo, por meio da disciplina de Padrões de Projeto de Sítios Internet II, o conhecimento foi ampliado com a introdução ao uso de *frameworks*. O projeto “Dolce Gelato”, um site fictício de uma sorveteria foi desenvolvido com o auxílio do *framework Tailwind* CSS, que utiliza classes utilitárias para aplicar estilos diretamente no HTML, permitindo a construção de interfaces responsivas e modernas com maior agilidade e organização do código.

No quarto capítulo, a disciplina de Programação de Sítios Internet III trouxe um aprofundamento na linguagem de programação *JavaScript*. Foi desenvolvido um sistema *Create, Read, Update, Delete* (CRUD) para gerenciamento de clientes e ordens de serviço de uma empresa fictícia de jardinagem. A aplicação utilizou conceitos como funções, vetores, objetos, manipulação do *Document Object Model* (DOM) e persistência de dados no navegador com o uso da *Application Programming Interface* (API) *LocalStorage*. O projeto reforçou o entendimento de lógica de programação, validação de dados, modularização do código e interatividade em aplicações web.

No quinto capítulo, a disciplina Banco de Dados e Internet II aprofundou os conhecimentos em acesso a bancos de dados, com foco na implementação de regras de negócio diretamente no Sistema de Gerenciamento de Banco de Dados (SGBD). Utilizou-se o *MySQL Workbench* para o desenvolvimento prático de um projeto inspirado em plataformas de *streaming*: o banco de dados bd_musicas que envolveu a modelagem de um esquema relacional completo, com entidades como usuários, músicas, artistas, álbuns e listas de reprodução, relacionadas através de chaves primárias e estrangeiras.

No sexto capítulo com a disciplina Tópicos Especiais em Sistemas para Internet II focamos no desenvolvimento de uma Biblioteca Online, uma aplicação web que permite a consulta, organização e download de livros digitais. O sistema conta com funcionalidades como favoritar livros, gerenciar uma lista de leitura e um sistema de autenticação de usuários. O projeto foi desenvolvido utilizando o *framework Angular* em conjunto com a biblioteca *Angular Material*.

No sétimo capítulo, com a disciplina de Desenvolvimento para Dispositivos Móveis II, foi implementado um aplicativo de Quiz utilizando a linguagem Java no Android Studio, o sistema permite ao usuário responder perguntas, navegar entre telas por meio de *Intents* e receber *feedback* sonoro imediato sobre erros ou acertos.

O projeto explorou conceitos essenciais de *Activities*, *MediaPlayer* e transferências de informações entre telas no ambiente Android.

2 PADRÕES DE PROJETO DE SITÍOS INTERNET I

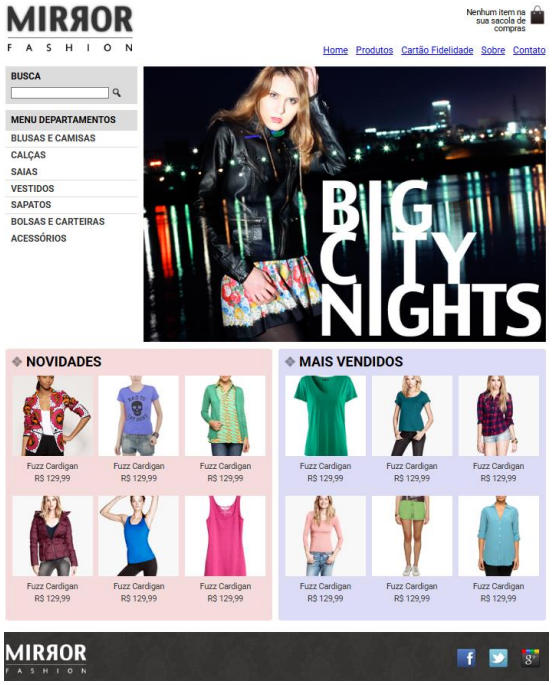
No decorrer do primeiro semestre do curso, a turma teve contato com a disciplina de Padrões de Projeto de Sítios e Internet ministrada pelo Prof. Dr. Adriano Bezerra, cujo objetivo principal foi introduzir o aluno ao básico de desenvolvimento web utilizando linguagem de marcação de texto HTML5 e folhas de estilo em cascata CSS por meio de um editor de código-fonte, o *Visual Studio Code* (VScode).

Ao longo das aulas foram abordados os princípios fundamentais do desenvolvimento onde foram aprendidas estruturas e estilização de páginas web, levando em consideração os conceitos básicos de usabilidade.

Como exercício prático foi desenvolvida a aplicação denominada “Mirror Fashion” um *E-commerce* no segmento de moda. O projeto Mirror Fashion foi desenvolvido com páginas estáticas, sem uso de linguagem de programação dinâmica como *JavaScript*, focando apenas em HTML e CSS com o intuito de exercer os conceitos aprendidos em aula e alcançar a compreensão do funcionamento básico da criação e estilização de páginas web.

A Figura 2.1 mostra a página inicial, nota-se que o layout da aplicação é divido visualmente por blocos, em termos de estrutura eles são implementados utilizando o HTML que é a base da criação de sítios web.

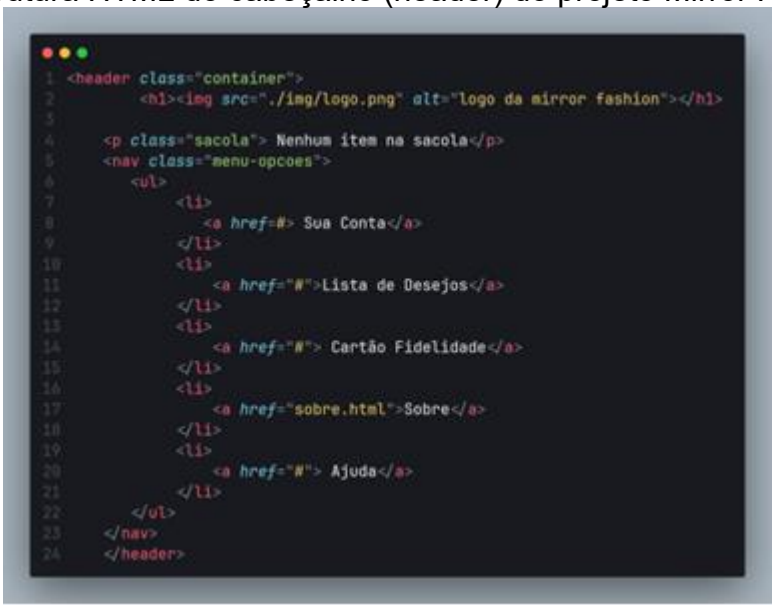
Figura 2.1 - Layout da página inicial (Homepage)



Fonte: Elaborada pela autora, 2023

A Figura 2.2 ilustra o cabeçalho da página utilizando a *tag* <*header*> que oferece aos usuários acesso rápido as páginas principais do site, como a logo e o menu de navegação. O menu foi estruturado com a *tag* <*nav*>, em conformidade com as recomendações de acessibilidade da *Web Content* Accessibility Guidelines (WCAG), que orientam a identificação clara das áreas de navegação para facilitar a intepretação por leitores de telas e outras tecnologias assistivas. Além disso, a marcação semântica com a *tag* <*ul*>, organizada em itens de lista <*li*>, garante que os elementos do menu sejam compreendidos como uma lista logica hierarquizada. O atendimento a essas diretrizes promove não apenas uma navegação mais inclusiva para pessoas com deficiência visual, motora ou cognitiva, mas também contribui para a robustez e a usabilidade geral do site (W3C, 2018).

Figura 2.2 - Estrutura HTML do cabeçalho (header) do projeto Mirror Fashion

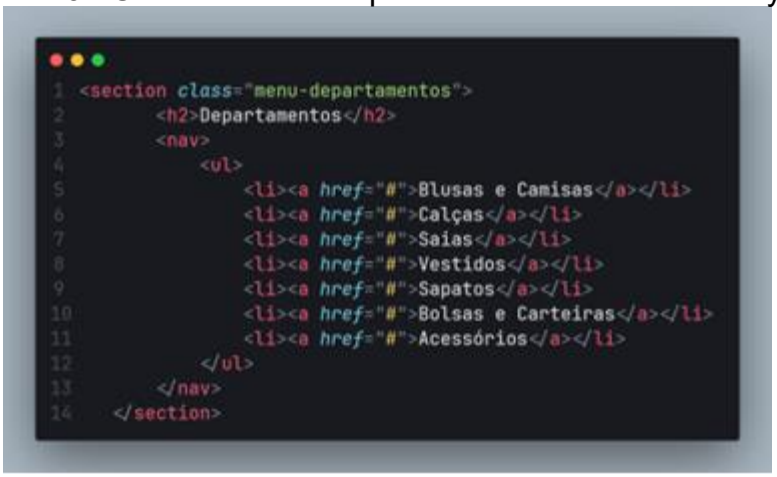


```
<header class="container">
        <h1><img src="./img/logo.png" alt="logo da mirror fashion"></h1>

    <p class="sacola"> Nenhum item na sacola</p>
    <nav class="menu-opcoes">
       <ul>
            <li>
               <a href=#> Sua Conta</a>
            </li>
            <li>
               <a href="#">Lista de Desejos</a>
            </li>
            <li>
               <a href="#"> Cartão Fidelidade</a>
            </li>
            <li>
               <a href="sobre.html">Sobre</a>
            </li>
            <li>
               <a href="#"> Ajuda</a>
            </li>
       </ul>
    </nav>
    </header>
```

Fonte: Elaborada pela autora, 2023

A divisão do layout em seções temáticas contribuiu para a organização lógica e padronizada do conteúdo, em alinhamento com as recomendações do HTML5. O bloco “Departamentos”, por exemplo, foi estruturado com a *tag* <*section*>, recurso que favorece a segmentação da página em áreas independentes, facilitando tanto a aplicação de estilos quanto a manutenção do código. Embora também colabore com a acessibilidade, seu principal papel é indicar que aquele conjunto de elementos possui um significado específico dentro da página. A Figura 2.3 ilustra essa estrutura, que se repete em outras áreas distintas do layout.

Figura 2.3 - Uso de <section> para divisão de blocos no layout



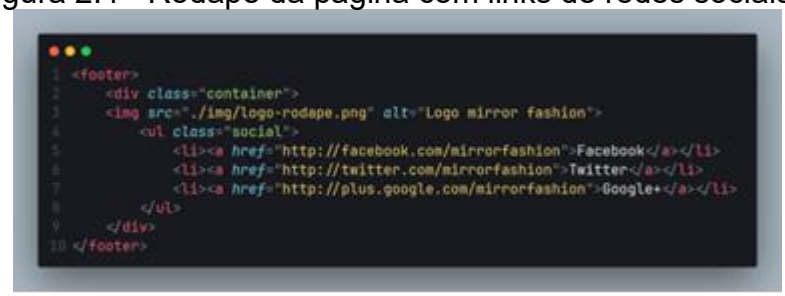
```
<section class="menu-departamentos">
        <h2>Departamentos</h2>
        <nav>
            <ul>
                <li><a href="#">Blusas e Camisas</a></li>
                <li><a href="#">Calças</a></li>
                <li><a href="#">Saias</a></li>
                <li><a href="#">Vestidos</a></li>
                <li><a href="#">Sapatos</a></li>
                <li><a href="#">Bolsas e Carteiras</a></li>
                <li><a href="#">Acessórios</a></li>
            </ul>
        </nav>
    </section>
```

Fonte: Elaborada pela autora, 2023

No rodapé foi utilizado a *tag <footer>* para grupar conteúdos complementares, como links para redes sociais e elementos da identidade visual. Esse recurso semântico do HTML5 permite concentrar informações relevantes, mas que não precisam ocupar espaço de destaque no corpo da página, mantendo-se acessíveis ao usuário ao final da navegação.

Além de contribuir para a consistência estrutural, o uso do *<footer>* reforça a hierarquia de informações e melhora a experiência de navegação.

Figura 2.4 - Rodapé da página com links de redes sociais



```
<footer>
    <div class="container">
    <img src="./img/logo-rodape.png" alt="Logo mirror fashion">
        <ul class="social">
            <li><a href="http://facebook.com/mirrorfashion">Facebook</a></li>
            <li><a href="http://twitter.com/mirrorfashion">Twitter</a></li>
            <li><a href="http://plus.google.com/mirrorfashion">Google+</a></li>
        </ul>
    </div>
</footer>
```

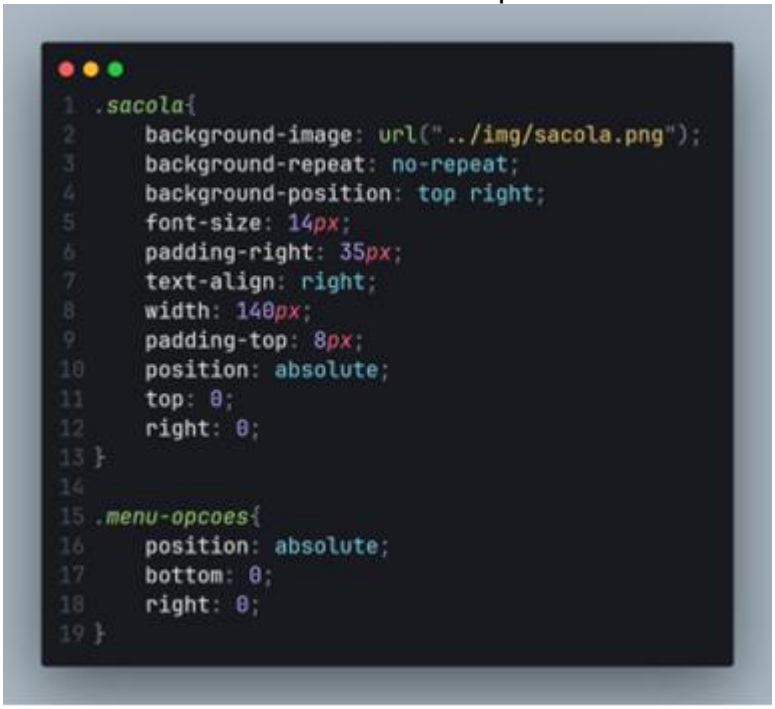
Fonte: Elaborada pela autora, 2023

A Figura 2.5 demonstra o uso de seletores bem definidos por classe (.sacola, .menu-opcoes) e elementos estruturais *(header, footer*), posicionamento estratégico usando *float, position, clear* e hierarquia visual estabelecida através de cores, espaçamentos e tamanhos da fonte.

O projeto implementou um sistema de layout baseado em *floats* e posicionamento absoluto, a estilização priorizou critérios como desempenho,

consciência visual e facilidade de manutenção. O CSS do projeto demonstra uma abordagem organizada, com estilos agrupados por secções da página, o posicionamento da sacola exemplifica a técnica de *image replacement* para elementos puramente decorativos (W3C, 2020).

Figura 2.5 - Trecho do código CSS responsável pelo posicionamento absoluto dos elementos 'sacola' e 'menu-opcoes'



```css
.sacola{
    background-image: url("../img/sacola.png");
    background-repeat: no-repeat;
    background-position: top right;
    font-size: 14px;
    padding-right: 35px;
    text-align: right;
    width: 140px;
    padding-top: 8px;
    position: absolute;
    top: 0;
    right: 0;
}

.menu-opcoes{
    position: absolute;
    bottom: 0;
    right: 0;
}
```

Fonte: Elaborada pela autora, 2023

A disciplina de Padrões de Projeto de Sítios Internet proporcionou uma base solida para o desenvolvimento web front-end. Por meio do projeto prático desenvolvido, foi possível aplicar os padrões web corretamente, criando páginas robustas e acessíveis. Essa experiencia prática consolidou os conhecimentos teóricos e serviu como ponto de partida para aprofundamento dos estudos na área.

3 PADRÕES DE PROJETO DE SÍTIOS INTERNET II

Durante o segundo semestre do curso, na disciplina de Padrões de Projeto de Sítios Internet II aplicada pelo Prof. Me. Felipe Maciel, foi desenvolvido um projeto prático com o objetivo de aplicar os conceitos de design responsivo, usabilidade, organização de código e aplicação de boas práticas no desenvolvimento de interfaces web.

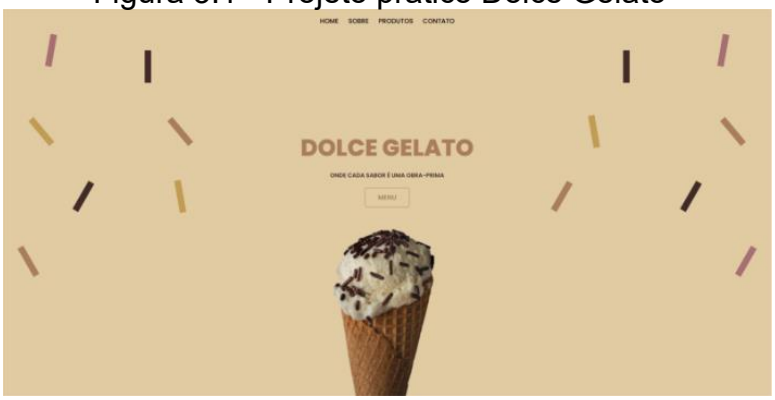
3.1 APRESENTAÇÃO DO PROJETO

O projeto consiste em uma sorveteria fictícia chamada Dolce Gelato, cujo site foi planejado para apresentar uma interface agradável, organizada e de fácil navegação. A estrutura do site foi construída com HTML, aliada à estilização feita exclusivamente em *Tailwind* CSS, que possibilitou o uso de classes utilitárias para acelerar o desenvolvimento e manter a consistência visual entre os componentes.

3.1.1 Tela inicial

A tela inicial do site apresenta o nome da sorveteria e um botão de navegação, com foco em atrair a atenção do visitante por meio de uma imagem destacada do produto e elementos visuais que remetem ao conceito da marca. A composição mostrada na figura 3.1 é simples e direta, valorizando a identidade visual e o apelo estético.

Figura 3.1 - Projeto prático Dolce Gelato

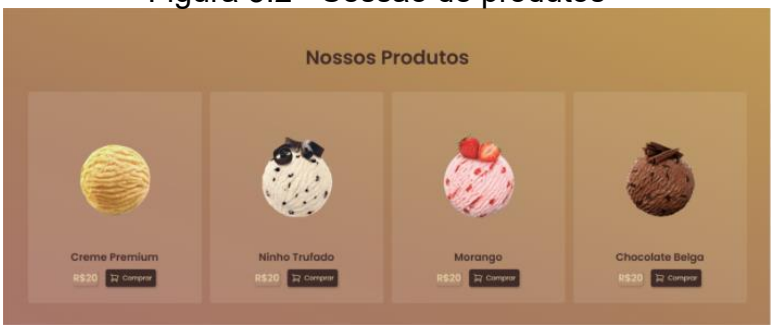


Fonte: Elaborada pela autora, 2023

3.1.2 Tela de produtos

A figura 3.2 exibe a sessão com os principais produtos da sorveteria em formato de cards, organizados horizontalmente. Cada *card* contém uma imagem, nome do sabor, preço e botão de compra, proporcionando ao usuário uma navegação intuitiva e direta, com ênfase na experiência de compra.

Figura 3.2 - Sessão de produtos



Fonte: Elaborada pela autora, 2023

3.1.3 Tela Sobre Nós

A tela “Sobre Nós” apresentada na figura 3.3 permite visualizar as características da sorveteria, destacando os diferenciais como uso de ingredientes naturais e sabores premium. Os ícones ao lado dos textos ajudam na leitura e na identificação visual das informações, reforçando a proposta de transparência e qualidade da marca.

Figura 3.3 - Sessão sobre as caracaterísticas da Dolce Gelato

Fonte: Elaborada pela autora, 2023

3.1.4 Registros Gelados e Rodapé

Por meio da figura 3.4 é possível visualizar a seção “Registros Gelados” que apresenta uma galeria de imagens que simula fotos reais dos produtos oferecidos pela sorveteria, com o objetivo de tornar a navegação mais atrativa e despertar o apetite visual do usuário. Logo abaixo, o rodapé do site disponibiliza informações institucionais, dados de contato e acesso às redes sociais, garantindo funcionalidade e completude à página.

Figura 3.4 - Sessão com uma galeria de imagens dos clientes

Fonte: Elaborada pela autora, 2023

Os principais objetivos foram criar uma interface atrativa e funcional, com forte ênfase na experiência do usuário, aplicando os padrões de projeto para web abordados em aula utilizando *frameworks*.

Os *frameworks* são estruturas reutilizáveis de código que oferecem uma base sólida para o desenvolvimento de aplicações, promovendo padronização, organização e economia de tempo. Lisboa (2008, p. 16) afirma:

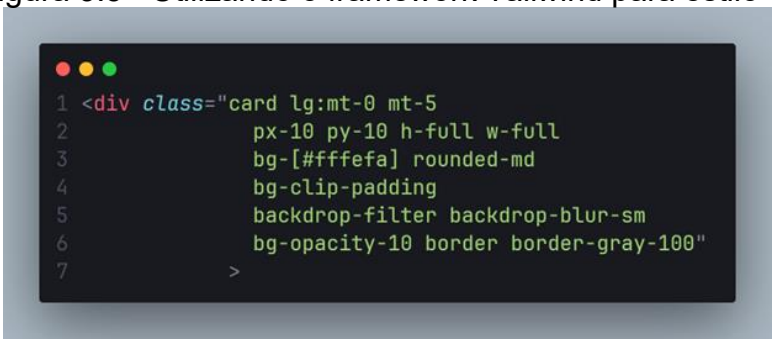
> A tradução da palavra *framework* não é muito clara. Assim prefiro utilizar a definição de Minetto (2007): um *framework* de desenvolvimento é uma ‘base de onde se pode desenvolver algo maior ou mais específico. É uma coleção de códigos-fonte, classes, funções, técnicas e metodologias que facilitam o desenvolvimento de novos softwares’.

3.2 UTILIZAÇÃO DO TAILWIND CSS

Nesse contexto a figura 3.5 demonstra o grande diferencial técnico do projeto que foi a adoção do *Tailwind CSS*, um *framework* de classes utilitárias para estilização, que permite aplicar estilos diretamente no HTML de forma padronizada, rápida e

organizada. Essa abordagem proporcionou maior produtividade, consistência visual e controle refinado sobre o layout da interface, facilitando a construção de componentes responsivos e modernos com mais agilidade e menos código *CSS* personalizado.

Figura 3.5 - Utilizando o framework Tailwind para estilo



Fonte: Elaborada pela autora, 2023

3.3 ANÁLISE TÉCNICA DO COMPONENTE

O trecho acima representa a estrutura de um card completo, responsável por agrupar visualmente os conteúdos internos (como nome do produto, preço, botão, imagem etc.)

Segundo a Mozilla (2025), o elemento *div* é um contêiner genérico para conteúdo de fluxo, sem semântica intrínseca, usado para agrupar elementos para estilização ou por compartilharem atributos, devendo ser utilizado apenas quando não houver um elemento semântico mais apropriado.

Tratando-se de uma *div* com diversas classes utilitárias que controlam o espaçamento, o layout, a responsividade e a estética visual do componente. A aplicação das classes mt-5 e lg:mt-0 garante que o espaçamento superior seja adaptável conforme o tamanho da tela, promovendo um comportamento responsivo. As classes px-10 py-10 adicionam espaçamento interno horizontal e vertical, conferindo conforto visual ao conteúdo interno do card. Com *w-full* e *h-full*, o card se ajusta à largura e altura disponíveis, preenchendo o espaço de forma proporcional.

A parte visual é cuidadosamente tratada com bg-[#fffefa], que define uma cor de fundo suave e personalizada. A classe *rounded-md* adiciona cantos levemente arredondados, contribuindo para um visual moderno e agradável. O destaque especial fica para o uso de efeitos visuais como *bg-opacity-10, backdrop-filter, backdrop-blur-sm e bg-clip-padding*, que em conjunto criam o chamado efeito de “*glassmorphism*”

uma aparência semelhante a vidro fosco, que traz sofisticação e leveza ao design. Por fim, a borda sutil aplicada com *border border-gray-100* finaliza o componente de maneira delicada.

3.4 DESIGN RESPONSIVO E MOBILE

Todo o layout foi desenvolvido de maneira responsiva, seguindo o princípio do design *mobile-first*, garantindo uma boa experiência de navegação em diferentes tamanhos de tela, como smartphones, tablets e desktops.

Figura 3.6 - Site Dolce Gelato responsivo na versão mobile

Fonte: Elaborada pela autora, 2023

A figura 3.6 mostra que com o uso do framework, esse princípio foi aplicado de forma prática e eficiente. *Tailwind* disponibiliza classes utilitárias específicas para diferentes pontos de quebra (*breakpoints*), permitindo que o layout se adapte dinamicamente conforme o tamanho da tela.

Por exemplo, classes como mt-5 (para dispositivos menores) e lg:mt-0 (para telas grandes) foram utilizadas para ajustar o espaçamento de elementos em diferentes resoluções. Essa abordagem *mobile-first* facilita a criação de interfaces flexíveis e acessíveis, sem a necessidade de media queries complexas, tornando o código mais limpo, intuitivo e produtivo para o desenvolvedor.

A experiência proporcionada por esse projeto contribuiu significativamente para a formação acadêmica e profissional, principalmente no que diz respeito à aplicação prática de padrões modernos de desenvolvimento front-end. Trabalhar com um framework como o *Tailwind CSS* possibilitou a construção de interfaces coesas, ao mesmo tempo em que proporcionou maior domínio sobre conceitos como

responsividade, modularidade e reutilização de código. Como afirma Jakob Nielsen (1999, p. 38), “a consistência é uma das maneiras mais poderosas de tornar os sites mais utilizáveis”, reforçando a importância das boas práticas seguidas ao longo do desenvolvimento do projeto Dolce Gelato.

4 PROGRAMAÇÃO DE SÍTIOS INTERNET III

No terceiro semestre do curso de Sistemas para Internet, foi lecionada a disciplina de Programação de Sítios Internet III apresentada pelo Prof. Dr. Everaldo Silva, cujo principal objetivo foi aprofundar o conhecimento em *JavaScript*.

Como forma de aplicar os conceitos teóricos estudados, foi desenvolvido um projeto prático de um sistema CRUD voltado à gestão de clientes e ordens de serviço de uma empresa fictícia de jardinagem.

Para armazenar todas essas informações localmente no navegador, utilizou-se a tecnologia *localStorage*:

> O Web Storage é uma especificação que fornece mecanismos para armazenar dados no navegador do usuário. Dentre eles, o localStorage permite que pares de chave-valor sejam armazenados de forma persistente, ou seja, os dados permanecem disponíveis mesmo após o fechamento do navegador. Esse tipo de armazenamento é limitado ao escopo do domínio, não sendo acessível por outras aplicações, e sua principal vantagem é a simplicidade de uso, sem a necessidade de interações com servidores ou bancos de dados. No entanto, deve ser usado com cautela, pois não é criptografado, possui limites de armazenamento e não deve conter informações sensíveis (Pilgrim, 2010, p. 147).

A aplicação permite que o usuário cadastre clientes com nome, telefone, e-mail e endereço, garantindo a validação de dados essenciais. Após o cadastro, é possível selecionar esses clientes para gerar ordens de serviço, escolhendo entre uma lista de serviços predefinidos como corte de grama, manutenção de jardins e instalação de irrigação. Cada ordem de serviço é associada a um cliente, data e lista de serviços prestados, sendo possível visualizar o custo total e detalhes da operação assim como mostra a figura 4.1

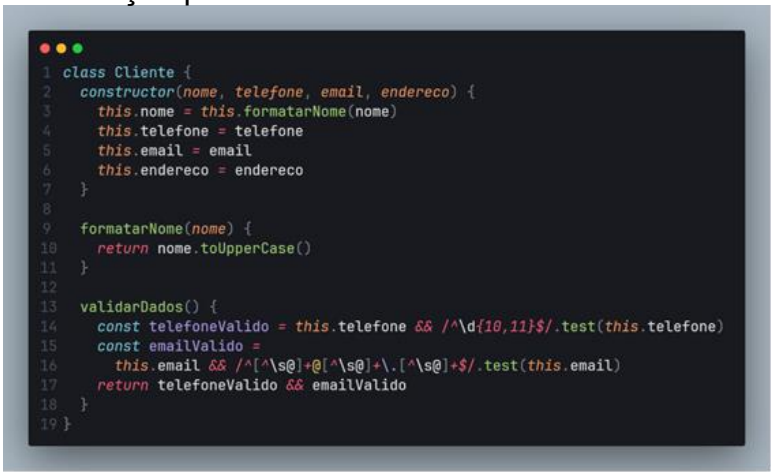
Figura 4.1 - Detalhes da operação

Fonte: Elaborada pela autora, 2024

4.1 ESTRUTURA DO CÓDIGO EM JAVASCRIPT

Durante o desenvolvimento do sistema, diversos conceitos da linguagem *JavaScript* foram aplicados, como declaração de variáveis, funções, vetores, objetos, estruturas de repetição e condicionais.

Figura 4.2 - Função para cadastrar cliente e armazenar no *localStorage*



```
class Cliente {
  constructor(nome, telefone, email, endereco) {
    this.nome = this.formatarNome(nome)
    this.telefone = telefone
    this.email = email
    this.endereco = endereco
  }

  formatarNome(nome) {
    return nome.toUpperCase()
  }

  validarDados() {
    const telefoneValido = this.telefone && /^\d{10,11}$/.test(this.telefone)
    const emailValido =
      this.email && /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(this.email)
    return telefoneValido && emailValido
  }
}
```

Fonte: Elaborada pela autora, 2024

No código na figura 4.2 o *script* tem como objetivo a manipulação da classe Cliente que representa um cliente da empresa e armazena informações básicas como nome, telefone, e-mail e endereço. Uma das decisões do projeto foi garantir que o nome do cliente sempre fosse armazenado em letras maiúsculas, de forma padronizada. Isso foi feito por meio do método formatarNome().

Além disso, a classe inclui um método validarDados() que verifica se o telefone informado possui entre 10 e 11 dígitos e se o e-mail possui um formato válido, utilizando expressões regulares.

4.2 FUNÇÕES

Durante o desenvolvimento do sistema, diversas funções foram implementadas para realizar ações específicas, como cadastrar clientes, gerar ordens de serviço, validar campos e exibir informações ao usuário.

De acordo com Grillo e Fortes (2008), no *JavaScript* as funções são tratadas como dados, podendo ser atribuídas a variáveis ou propriedades de objetos e

utilizadas normalmente. Além disso, podem ser passadas como parâmetros para outras funções ou até mesmo retornados por elas, motivo pelo qual são conhecidas como funções de alta ordem.

Na imagem 4.3 pode-se observar uma função utilizada para exibir os detalhes de uma ordem de serviço específica, com base no índice do *array* que a armazena.

Figura 4.3 - Função para exibir detalhes da ordem de serviço

```
const exibirDetalhesOrdem = (index) => {
  const ordem = ORDENS_DE_SERVICO[index]
  const detalhes = `Cliente: ${ordem.cliente.nome}\nData: ${
    ordem.data
  }\nServiços:\n${ordem.servicos
    .map((s) => `${s.nome} - R$${s.custo.toFixed(2)}`)
    .join("\n")}\nCusto Total: R$${ordem.servicos
    .reduce((total, servico) => total + servico.custo, 0)
    .toFixed(2)}`
  alert(detalhes)
}
```

Fonte: Elaborada pela autora, 2024

Essa função foi escrita na forma de uma função anônima atribuída a uma constante (*arrow function*). Ela recebe como parâmetro o index, que representa a posição da ordem dentro do *array* ORDENS_DE_SERVICO.

Dentro da função, são realizadas as seguintes operações: Acesso ao objeto da ordem usando o índice informado; criação de uma *string* com os dados formatados: nome do cliente, data, lista de serviços com valores individuais e o cálculo do valor total da ordem; exibição dessas informações ao usuário utilizando o método alert() do navegador.

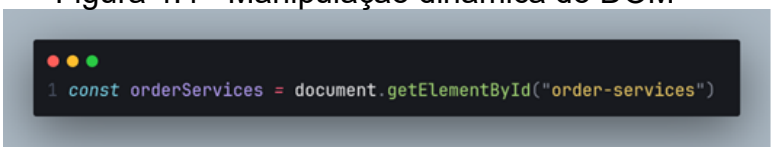
Para montar a lista de serviços, utilizou-se o método map(), enquanto o custo total foi calculado com o método reduce(), ambos aplicados sobre o *array* de serviços daquela ordem.

4.3 MANIPULAÇÃO DO DOCUMENT OBJECT MODEL (DOM)

O *Document Object Model (DOM)* é uma interface de programação que representa a estrutura do documento HTML como uma árvore de objetos, permitindo que linguagens de script, como o *JavaScript*, acessem, modifiquem e controlem o conteúdo, estrutura e estilo da página web em tempo real.

Na implementação, foram utilizados métodos do DOM para capturar elementos específicos do HTML, como menus de seleção e listas, bem como para criar elementos novos dinamicamente, como opções de serviços e itens de lista de clientes. Além disso, foram aplicados eventos para capturar as ações do usuário, como o envio de formulários e cliques em botões, possibilitando o tratamento adequado dos dados e a atualização visual imediata da interface, sem a necessidade de recarregar a página.

Figura 4.4 - Manipulação dinâmica do DOM



Fonte: Elaborada pela autora, 2024

Neste trecho da figura 4.4, o código utiliza o DOM para acessar o elemento HTML que possui o atributo id="order-services" que, neste caso, corresponde a um elemento <select>. Essa referência permite que o script manipule diretamente esse elemento na página, adicionando opções (*<option>*) de forma dinâmica, ou seja, durante a execução do programa, sem a necessidade de alterar manualmente o código HTML estático.

4.4 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Durante o processo de desenvolvimento do sistema, foram observados os seguintes requisitos funcionais:

- Os usuários devem ser capazes de cadastrar novos clientes, informando nome, telefone, e-mail e endereço;
- O sistema deve validar os dados inseridos, como e-mail e telefone, antes de permitir o cadastro;
- Os usuários devem visualizar a lista de clientes cadastrados;
- Os usuários devem ser capazes de remover clientes da base de dados local;
- Os usuários devem ser capazes de cadastrar ordens de serviço, associando-as a um cliente já existente;

- O sistema deve permitir a inserção de múltiplos serviços em uma única ordem;
- O sistema deve calcular automaticamente o custo total da ordem de serviço;
- Os usuários devem visualizar uma lista de ordens de serviço criadas;
- Os usuários devem ser capazes de excluir ordens de serviço;
- O sistema deve exibir um resumo com o nome do cliente, data da ordem, serviços incluídos e valor total.

E os requisitos não funcionais são:

- O sistema deve funcionar completamente offline, sem necessidade de conexão com a internet;
- O sistema deve persistir os dados localmente utilizando a API *localStorage*, mesmo após o navegador ser fechado;
- O aplicativo deve possuir uma interface simples, acessível e intuitiva para usuários com pouco conhecimento técnico

4.5 PERSISTÊNCIA DE DADOS COM LOCALSTORAGE

A persistência de dados é um aspecto fundamental em aplicações web que visam manter informações armazenadas mesmo após o fechamento ou recarregamento da página. No projeto desenvolvido, essa necessidade foi atendida por meio do uso da *Web Storage API*, mais especificamente da funcionalidade *LocalStorage*, fornecida pelos navegadores modernos.

No projeto, foi utilizado para armazenar duas entidades principais: os clientes e as ordens de serviço. Para isso, ao adicionar ou remover elementos dessas listas, os dados são serializados usando JSON.stringify() e gravados com localStorage.setItem(). Posteriormente, ao carregar a aplicação, esses dados são recuperados com localStorage.getItem() e desserializados com JSON.parse(), como ilustrado na figura 4.5

Figura 4.5 - Utilização do *LocalStorage*

```
const ORDENS_DE_SERVICO =
  JSON.parse(localStorage.getItem("ORDENS_DE_SERVICO")) || []
const CLIENTES = JSON.parse(localStorage.getItem("CLIENTES")) || []
```

Fonte: Elaborada pela autora, 2024

O *LocalStorage* é uma ferramenta essencial para desenvolvedores web, permitindo o armazenamento de dados persistentes no navegador do usuário. Essa funcionalidade abre um leque de possibilidades para suas aplicações, desde a personalização da experiência do usuário até a persistência de dados de sessão.

Além disso, por se tratar de uma aplicação voltada para fins acadêmicos e de aprendizagem, o uso oferece uma ótima oportunidade para compreender conceitos de persistência, serialização de objetos, e armazenamento estruturado no lado do cliente.

5 BANCO DE DADOS E INTERNET II

A disciplina Banco de Dados e Internet II foi instruída pelo Prof. Dr. Alciano Oliveira e teve como propósito aprofundar os conhecimentos adquiridos anteriormente e explorar recursos avançados no acesso a banco de dados. Seu objetivo geral foi capacitar o aluno na utilização de técnicas que envolvessem a implementação das regras de negócio diretamente no banco de dados.

5.1 MODELO ENTIDADE-RELACIONAMENTO E CONCEITOS

Foram abordados os diagramas de Modelo de Entidade e Relacionamento (MER).

> O modelo de dados entidade-relacionamento (E-R) tem por base a percepção do mundo real como um conjunto de objetos básicos, chamados entidades, e do relacionamento entre eles. Uma entidade é uma "'coisa" ou um "objeto" do mundo real que pode ser identificado por outros objetos. As entidades são descritas no banco de dados por meio de seus atributos. O conjunto de todas as entidades de um mesmo tipo, assim como o conjunto de todos os relacionamentos de mesmo tipo são denominados conjunto de entidades e conjunto de relacionamentos, respectivamente. (A. Silberschatz, H. F. Korth, and S. Sudarshan, 2006, p. 7)

Os estudos foram desenvolvidos com base na ferramenta MySQL Workbench, um ambiente visual fornecido pela Oracle para criação, modelagem e manipulação de bancos de dados relacionais. Através desta plataforma, foi possível criar comandos, estruturar tabelas, fazer relacionamentos e executar consultas utilizando a linguagem *Structured Query Language* (*SQL*).

Segundo a Apostila de Banco de Dados e SQL (s.d.), quando os bancos de dados relacionais estavam sendo desenvolvidos, foram criadas linguagens para sua manipulação. O Departamento de Pesquisas da IBM desenvolveu a SQL como interface para o sistema de BD relacional SYSTEM R, no início da década de 1970. Em 1986, o *American National Standards Institute* (ANSI) publicou um padrão SQL, estabelecendo-a como linguagem padrão para bancos de dados relacionais.

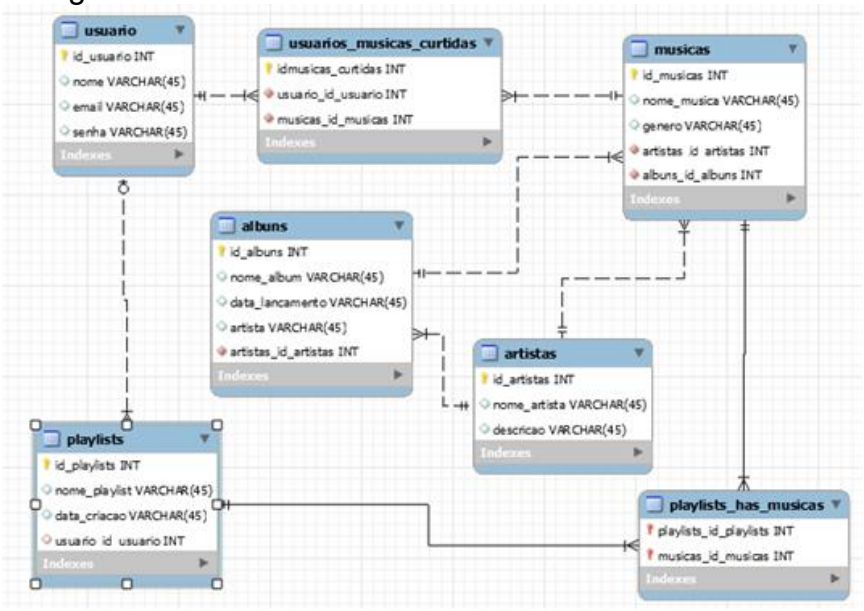
5.2 PROJETO PRÁTICO DO BANCO DE MÚSICAS

O projeto escolhido para aplicação prática dos conteúdos foi o banco de dados bd_musicas, uma proposta inspirada em plataformas de streaming. Esse banco foi

idealizado para armazenar informações de usuários, músicas, álbuns, artistas e playlists, permitindo a simulação de um sistema de gerenciamento musical.

A Figura 5.1 apresenta um exemplo prático do modelo de dados desenvolvido para o banco, a partir desse modelo, foi possível estruturar um banco de dados completo e, na sequência, realizar comandos que simularam a manipulação de dados reais.

Figura 5.1 - Modelo de Entidade e Relacionamento



Fonte: Elaborada pela autora, 2024

Esse modelo segue o padrão relacional, no qual cada tabela possui uma chave primária, responsável por identificar de forma única cada registro armazenado. Por exemplo, a tabela usuario possui o campo id_usuario, utilizado para distinguir os diferentes usuários cadastrados no sistema. Da mesma forma, entidades como músicas, artistas, albuns e playlists também têm suas próprias chaves primárias, como id_musicas, id_artistas, id_albuns e id_playlists, respectivamente.

Um dos pontos mais importantes no modelo relacional é o relacionamento entre tabelas, feito através de chaves estrangeiras. Essas chaves permitem que as tabelas se comuniquem entre si, estabelecendo vínculos lógicos entre os dados. Um exemplo claro disso é a tabela albuns, que contém o campo artistas_id_artistas como chave estrangeira, conectando cada álbum a seu respectivo artista.

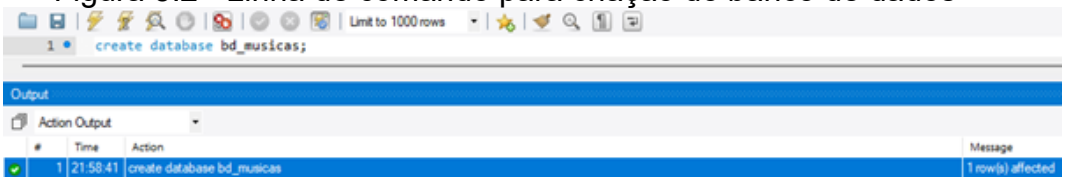
Já a tabela playlists possui a coluna usuario_id_usuario, que relaciona cada playlist ao usuário que a criou. Além dos relacionamentos um-para-muitos, o banco

também implementa relacionamentos muitos-para-muitos por meio de tabelas auxiliares. A tabela playlists_has_musicas é um exemplo disso: ela conecta músicas a playlists, permitindo que uma mesma música esteja presente em várias playlists, e que cada playlist contenha várias músicas. Essa estrutura é composta por duas chaves estrangeiras: playlists_id_playlists e musicas_id_musicas.

Outro caso semelhante é a tabela usuarios_musicas_curtidas, que registra quais músicas foram curtidas por quais usuários, contribuindo para funcionalidades de personalização, estatísticas e recomendações.

A criação e o acesso a um banco de dados no MySQL são feitos por meio de comandos específicos da linguagem SQL, conforme ilustrado na figura 5.2, é possível compreender um dos principais nesse processo, o comando CREATE DATABASE.

Figura 5.2 - Linha de comando para criação do banco de dados



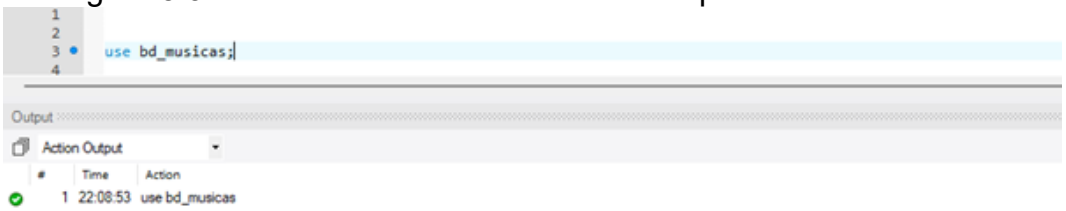
Fonte: Elaborada pela autora, 2024

Segundo o DATASUS (2025), um Sistema de Gerenciamento de Banco de Dados (SGBD) é o conjunto de programas de computador responsáveis pelo gerenciamento de bases de dados. O principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, manipulação e organização dos dados. O SGBD disponibiliza uma interface para que os seus clientes possam incluir, alterar ou consultar dados.

O comando CREATE DATABASE é utilizado para criar um banco de dados dentro do SGBD. No contexto do projeto bd_musicas, esse comando foi responsável por iniciar a estrutura lógica que armazenaria todas as tabelas, relacionamentos e dados do sistema.

Já o comando USE demonstrado na figura 5.3 tem como função selecionar qual banco de dados será utilizado para as próximas operações. Após criar o banco de dados, é necessário indicá-lo explicitamente para que o SGBD entenda onde as instruções subsequentes devem ser executadas.

Figura 5.3 - Selecionando o banco de dados para ser utilizado



Fonte: Elaborada pela autora, 2024

A partir desse comando, todas as tabelas, inserções, atualizações ou consultas feitas por meio de instruções SQL afetarão o banco bd_musicas. Essa etapa é essencial no processo de desenvolvimento, pois garante que todas as operações realizadas estejam vinculadas à base de dados correta, evitando erros e inconsistências na manipulação dos dados.

5.3 ESTRUTURAÇÃO DAS TABELAS

Durante o desenvolvimento do projeto, foi amplamente utilizada a Linguagem de Definição de Dados (DDL). Essa linguagem, que faz parte da SQL é responsável pela criação, alteração e exclusão de estruturas dentro de um banco de dados relacional.

O comando CREATE TABLE foi utilizado para definir a estrutura de uma tabela dentro do banco criado. Esse comando permite declarar os nomes das colunas (atributos), seus respectivos tipos de dados, restrições e relacionamentos com outras tabelas. No contexto do projeto, cada entidade do sistema foi representada por uma tabela distinta, cada tabela foi criada utilizando o comando CREATE TABLE, com definição clara de sua chave primária (PRIMARY KEY), chaves estrangeiras (FOREIGN KEY) e os tipos de dados apropriados para cada coluna.

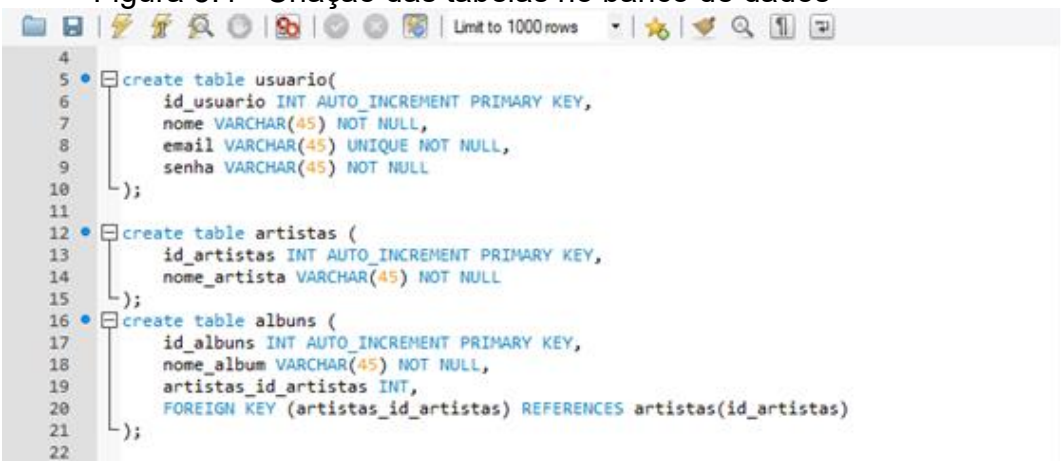
Na imagem 5.4 podemos observar A palavra INT (do inglês *integer*) define um tipo de dado numérico inteiro. É utilizado para armazenar números sem casas decimais. No banco bd_musicas, o INT foi usado para representar identificadores únicos, esse campo armazena o número de identificação de cada usuário, sendo uma chave primária.

O modificador AUTO_INCREMENT é usado junto com campos do tipo INT, normalmente em chaves primárias, para que o valor desse campo seja gerado automaticamente a cada novo registro inserido. Nesse caso, o banco de dados vai

preencher automaticamente o campo id_artistas com um número sequencial (1, 2, 3...) sempre que um novo artista for adicionado.

VARCHAR é um tipo de dado que armazena cadeias de texto (*strings*) com tamanho variável, o número entre parênteses define o limite máximo de caracteres. A cláusula NOT NULL define que o campo não pode ficar vazio e isso garante que o dado é obrigatório no momento da inserção de um novo registro e a restrição UNIQUE é usada para garantir que não haja valores repetidos nesse campo em toda a tabela, por meio da figura 5.4 é possível visualizar a criação das tabelas no banco de dados.

Figura 5.4 - Criação das tabelas no banco de dados



Fonte: Elaborada pela autora, 2024

5.4 MANIPULAÇÃO DE DADOS

Para fazer inserção de registros utilizamos o comando INSERT INTO que é fundamental para alimentar as tabelas com os dados necessários, sua sintaxe permite especificar quais colunas devem ser preenchidas e quais valores devem ser atribuídos a essas colunas no momento da inserção.

A utilização do comando INSERT INTO para adicionar múltiplos registros em uma única instrução SQL. Essa é uma prática eficiente que reduz a quantidade de comandos executados e melhora o desempenho do banco de dados, especialmente durante o preenchimento inicial de tabelas.

A primeira instrução mostrada na imagem refere-se à tabela usuario, neste exemplo, dois usuários são inseridos de uma só vez, cada linha dentro da cláusula VALUES representa um novo registro, com os campos nome, email e senha

preenchidos. O campo id_usuario, por ser AUTO_INCREMENT, será atribuído automaticamente pelo banco para cada novo usuário.

A segunda instrução insere dois registros na tabela artistas, os dados dos artistas estão sendo armazenados com nome e uma breve descrição, essa tabela é importante para manter as referências das músicas e álbuns dentro do sistema. A figura 5.5 demonstra a utilização do INSERT INTO dessa forma evidencia o domínio do comando em sua forma mais otimizada, permitindo a inserção de diversos dados com clareza, economia de código e melhor desempenho durante a carga inicial do banco de dados.

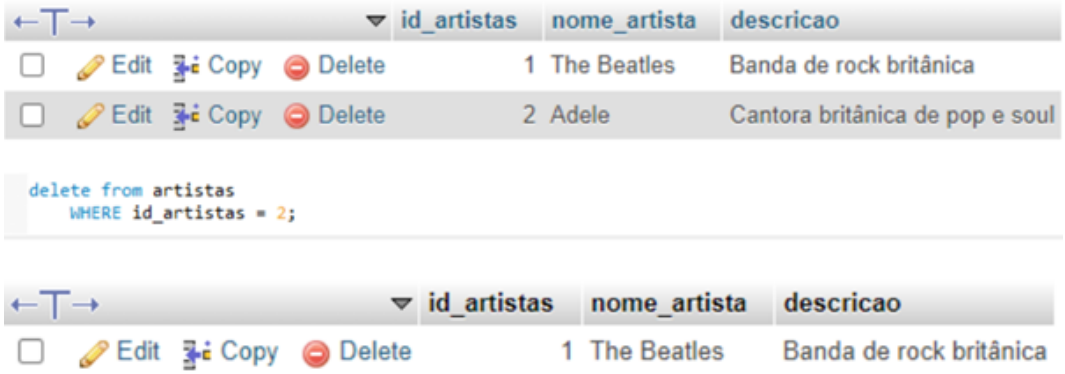
Figura 5.5 - Inserção de registros no banco de dados

```
insert into usuario (nome, email, senha)
    VALUES ('João', 'joao.zinho@email.com', 'senha123'),
           ('Maria', 'maria.zinha@email.com','senha321');

insert into artistas (nome_artista, descricao)
    VALUES ('The Beatles', 'Banda de rock britânica'),
           ('Adele', 'Cantora britânica de pop e soul');
```

Fonte: Elaborada pela autora, 2024

O comando DELETE ilustrado na figura 5.6 é utilizado para remover registros de uma tabela no banco de dados, e deve ser aplicada com atenção, pois uma vez executada, os dados são eliminados permanentemente (a menos que exista algum tipo de backup ou sistema de recuperação).

Figura 5.6 - Utilização do comando delete no banco de dados



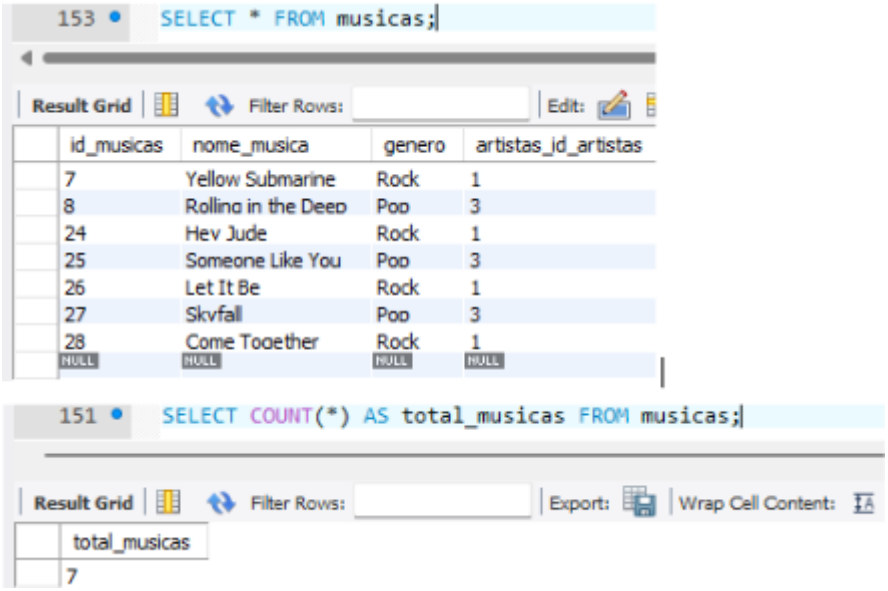
Fonte: Elaborada pela autora, 2024

Esse comando remove da tabela usuario o registro cujo id_usuario seja igual a 3, é importante observar que o uso da cláusula WHERE é essencial para especificar qual registro deve ser removido, caso ela seja omitida, todos os dados da tabela poderão ser excluídos. Para validar a remoção dos dados, foi utilizado o XAMPP, por meio do painel phpMyAdmin. Após executar o comando DELETE, foi possível atualizar a visualização da tabela no phpMyAdmin e confirmar que o registro especificado havia sido efetivamente excluído do banco de dados.

Apesar de o SELECT não modificar os dados (como INSERT, UPDATE ou DELETE), ele manipula os dados no sentido de acessá-los, filtrá-los e visualizá-los. Isso o coloca dentro do grupo de comandos *Data Manipulation Language* (DML). Quando se deseja exibir um único registro específico, a instrução SELECT é combinada com a cláusula WHERE, que filtra os resultados com base em uma condição, geralmente utilizando a chave primária da tabela, e a cláusula FROM é utilizada para listar todos os registros e todas as colunas, bastante utilizado para visualização geral da tabela.

No exemplo da figura 5.7, O segundo comando faz uso da função agregada COUNT (*), que retorna à quantidade total de registros existentes na tabela musicas. no caso exibido, o resultado foi 7, indicando que há sete músicas cadastradas. O uso de AS total_musicas atribui um nome mais legível à coluna do resultado da contagem.

Figura 5.7 - Contagem das linhas de registro

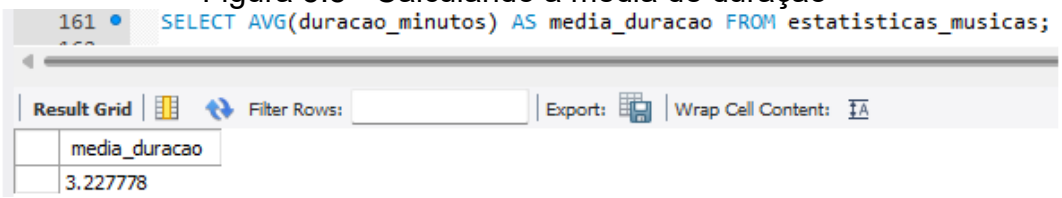


Fonte: Elaborada pela autora, 2024

De acordo com *O Modelo Relacional* (MACÁRIO; BALDO, 2005), “o símbolo ‘*’ na consulta indica que a relação resultante deve conter todos os atributos existentes na relação consultada”.

A instrução apresentada na figura 5.8 utiliza a função agregada AVG () para calcular a média da duração das músicas armazenadas na tabela estatisticas_musicas. O campo duracao_minutos representa a duração de cada música em minutos.

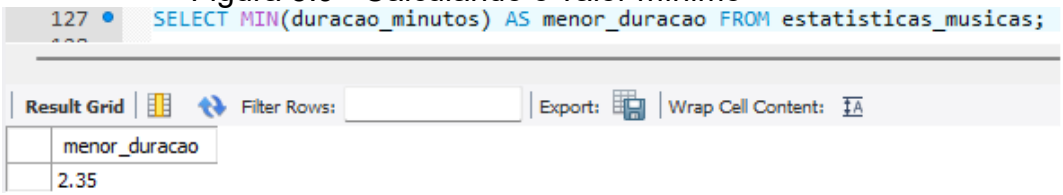
Figura 5.8 - Calculando a média de duração



Fonte: Elaborada pela autora, 2024

Na figura 5.9 é utilizada a função MIN (duracao_minutos) foi aplicada à tabela estatisticas_musicas com o objetivo de identificar a menor duração entre todas as músicas cadastradas.

Figura 5.9 - Calculando o valor mínimo



Fonte: Elaborada pela autora, 2024

A disciplina proporcionou uma base sólida na construção e manipulação de bancos de dados relacionais, reforçando a importância da linguagem SQL no desenvolvimento de sistemas, por meio da criação do projeto mencionado, foi possível aplicar comandos da DDL e da DML para estruturar, alimentar e consultar os dados de forma prática permitindo não apenas consolidar o conteúdo teórico, mas também vivenciar situações reais de desenvolvimento e administração de banco de dados.

6 TÓPICOS ESPECIAIS EM SISTEMAS PARA INTERNET II

Neste capítulo, será apresentado o desenvolvimento do projeto Biblioteca Online feito durante a disciplina de Tópicos Especiais em Sistemas para Internet II ministrada pelo Prof. Dr. Everaldo Silva, onde o objetivo principal do sistema é proporcionar uma plataforma acessível e intuitiva para consulta, organização e gerenciamento de livros em formato digital, com funcionalidades como favoritar livros, selecionar para que possam ser baixados futuramente, sendo essas funcionalidades utilizados apenas por usuários que estão ativos, a biblioteca conta com uma tela de login e cadastro para novos usuários.

O desenvolvimento utilizou as tecnologias Angular e Angular Material para construção da interface, além dos conhecimentos em HTML, CSS e *TypeScript* aplicados para a implementação das funcionalidades, organização do layout e interatividade.

6.1 TECNOLOGIAS E CONCEITOS

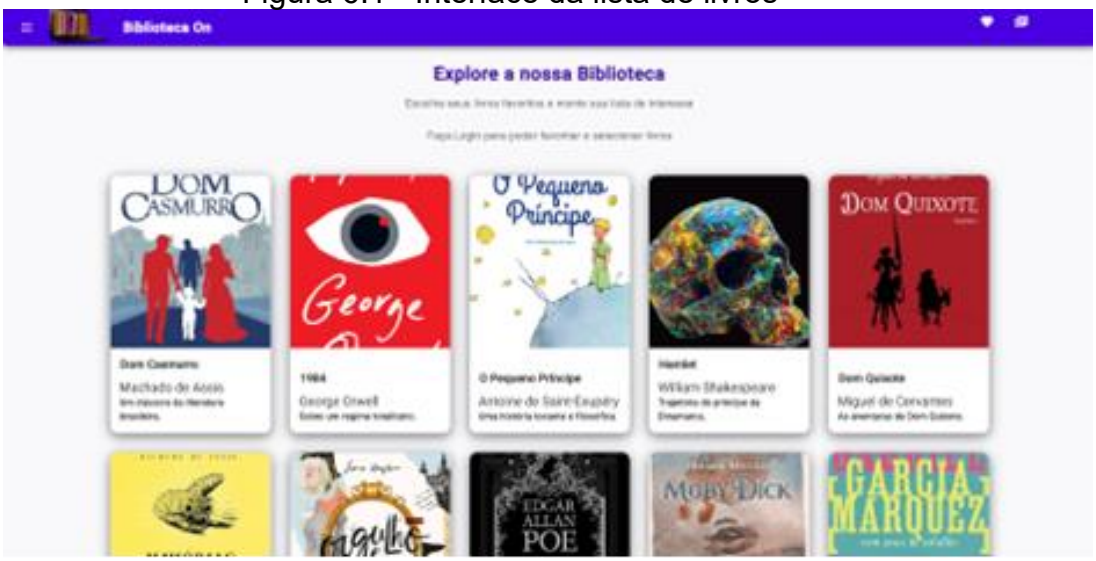
Angular é um framework *open-source* para desenvolvimento de aplicações web que permite criar *Single Page Applications* (SPA) utilizando *TypeScript*. Sua arquitetura baseada em componentes facilita a organização, reutilização e manutenção do código (Freeman e Robson, 2020).

Angular Material é uma biblioteca de componentes UI para Angular que implementa o Material Design do Google, oferecendo uma coleção de elementos visuais responsivos, acessíveis e customizáveis (Wargo, 2019).

TypeScript é uma linguagem de programação *open-source* desenvolvida pela Microsoft que estende o *JavaScript* ao adicionar tipagem estática opcional. Isso permite que os desenvolvedores detectem erros em tempo de compilação, melhorando a qualidade e a manutenção do código (Bierman, 2014).

A figura 6.1 exibe a interface de listagem dos livros disponíveis na Biblioteca Online. Cada livro é apresentado em um card, que contém uma imagem da capa, título, autor e uma breve descrição. Essa estrutura foi implementada utilizando o componente mat-card do Angular Material.

Figura 6.1 - Interface da lista de livros

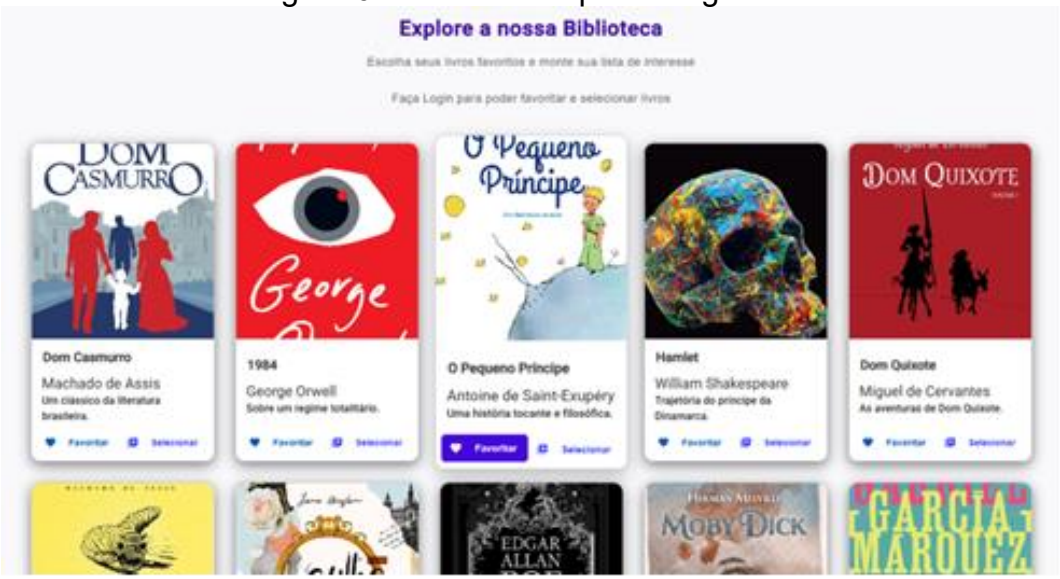


Fonte: Elaborada pela autora, 2025

Importante destacar que as ações disponíveis como “Favoritar” e “Selecionar” são exibidas somente para usuários autenticados. Essa lógica é implementada na camada de *template* com a diretiva **ngIf* que verifica, por meio do método authService.isLoggedIn(), se o usuário está logado.

Dessa forma, o sistema garante que apenas usuários com sessão ativa possam interagir com funcionalidades que modificam seus dados. A figura 6.2 mostra a mudança na interface após login do usuário.

Figura 6.2 - Interface após o Login



Fonte: Elaborada pela autora, 2025

6.2 CONTROLE DE ACESSO E AUTENTICAÇÃO

No Angular, a diretiva estrutural **ngIf* é utilizada para controlar a renderização condicional de elementos HTML com base em expressões booleanas. Isso significa que um elemento só será inserido no DOM se a condição especificada for verdadeira.

Por meio da figura 6.3 e possível compreender o serviço authService, uma estrutura criada para centralizar a lógica de autenticação no sistema, realiza validações e protege rotas ou funcionalidades com base na autenticação. No contexto da aplicação, esse serviço contém o método isLoggedIn(), responsável por verificar se há um usuário ativo com sessão iniciada.

Figura 6.3 - Método *authService*

```
<mat-card-actions *ngIf="authService.isLoggedIn()">
        <button mat-button color="primary" (click)="favoritarLivro(livro)">
          <mat-icon>favorite</mat-icon>
          Favoritar
        </button>
        <button mat-button color="accent" (click)="selecionarLivro(livro)">
          <mat-icon>library_add</mat-icon>
          Selecionar</button>
      </mat-card-actions>
```

Fonte: Elaborada pela autora, 2025

A função favoritarLivro(livro) verifica se o livro já foi adicionado à lista de favoritos. Caso não esteja, ele é incluído no array favoritos[ ].

Caso já esteja, o sistema exibe uma mensagem avisando o usuário, esse controle evita duplicatas, o armazenamento acontece em memória (no array favoritos) durante a sessão atual.

A figura 6.4 detalha a lógica desse método inicia com uma verificação de autenticação, o sistema prossegue com a próxima etapa: verificar se o livro que está sendo favoritado já foi adicionado anteriormente. Isso é feito utilizando o método *find*() sobre o *array* favoritos, que armazena todos os livros favoritados pelo usuário durante a sessão.

Figura 6.4 - Método favoritarLivro

```
favoritarLivro(livro: any) {
    if (!this.authService.isLoggedIn()) {
      alert('Você precisa estar logado para favoritar um livro.');
      return;
    }

    const jaFavoritado = this.favoritos.find(f => f.titulo === livro.titulo);
    if (!jaFavoritado) {
      this.favoritos.push(livro);
      alert(`Livro favoritado: ${livro.titulo}`);
    } else {
      alert(`O livro "${livro.titulo}" já está nos favoritos.`);
    }
  }
```

Fonte: Elaborada pela autora, 2025

A função *find*() percorre os elementos do array e retorna o primeiro item que tenha o mesmo título do livro atual. Caso esse retorno seja *undefined*, o sistema adiciona o novo livro à lista utilizando o método *push*(), e exibe uma nova mensagem informando que o livro foi favoritado com sucesso.

Outra funcionalidade essencial da Biblioteca Online é a possibilidade de o usuário selecionar livros para leitura posterior. Essa ação é implementada no método selecionarLivro(livro), que segue uma estrutura lógica bastante semelhante ao método de favoritar livros.

Se o usuário estiver autenticado, o método segue para verificar se o livro que está sendo selecionado já foi adicionado anteriormente à lista de leitura.

Essa verificação é feita por meio do método *find*() aplicado ao *array* listaDeLeitura, que armazena os livros escolhidos pelo usuário. O *find*() percorre os elementos da lista e busca um objeto cujo atributo titulo seja igual ao título do livro atual.

Caso o livro ainda não esteja na lista, ele é adicionado ao *array* listaDeLeitura com o uso do método *push*(). Conforme ilustrado na figura 6.5, o sistema exibe uma mensagem de confirmação para o usuário, indicando que o livro foi adicionado com sucesso.

Figura 6.5 - Método SelecionarLivro

```
selecionarLivro(livro: any) {
  if (!this.authService.isLoggedIn()) {
    alert('Você precisa estar logado para adicionar à lista de leitura.');
    return;
  }

  const jaSelecionado = this.listaDeLeitura.find(l => l.titulo === livro.titulo);
  if (!jaSelecionado) {
    this.listaDeLeitura.push(livro);
    alert(`Livro adicionado à lista de leitura: ${livro.titulo}`);
  } else {
    alert(`O livro "${livro.titulo}" já está na lista de leitura.`);
  }
}
```

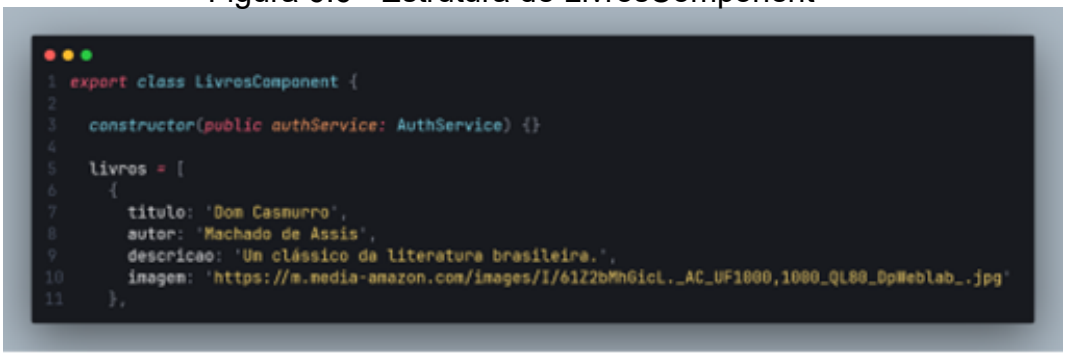
Fonte: Elaborada pela autora, 2025

6.3 DECLARAÇÃO DO CONSTRUTOR E VETORES

A estrutura do componente LivrosComponent começa com a declaração de um construtor, que tem como objetivo principal injetar dependências necessárias para o funcionamento do componente. No caso da Biblioteca Online, o construtor recebe como parâmetro o serviço *AuthService*, responsável por gerenciar o estado de autenticação do usuário.

Segundo a figura 6.6, é possível compreender o componente que define internamente um vetor chamado livros, que contém uma lista pré-definida de objetos com dados representando diferentes obras literárias. Cada objeto no vetor inclui atributos como título, autor, descrição e imagem. Esses dados são utilizados na interface para renderizar os cards que representam visualmente cada livro.

Figura 6.6 - Estrutura de LivrosComponent

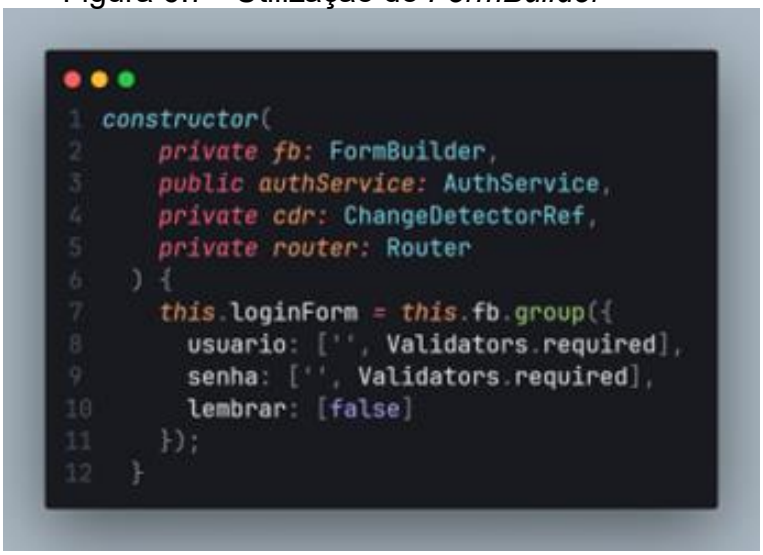


```
export class LivrosComponent {

  constructor(public authService: AuthService) {}

  livros = [
    {
      titulo: 'Dom Casmurro',
      autor: 'Machado de Assis',
      descricao: 'Um clássico da literatura brasileira.',
      imagem: 'https://m.media-amazon.com/images/I/6122bMhGicL._AC_UF1000,1000_QL80_DpWeblab_.jpg'
    },
```

Fonte: Elaborada pela autora, 2025

A aplicação da Biblioteca Online possui um sistema de login, no construtor do componente apresentado na figura 6.7, observa-se a utilização do serviço *FormBuilder*, que facilita a criação e o gerenciamento do formulário reativo (*FormGroup*). O formulário de login é inicializado com três campos: usuário, senha e lembrar, sendo os dois primeiros validados como obrigatórios através da diretiva *Validators.required*.

Figura 6.7 - Utilização do *FormBuilder*



Fonte: Elaborada pela autora, 2025

6.4 RENDERIZAÇÃO CONDICIONAL COM NGIF

Na aplicação, a "Lista de Leitura" é apresentada ao usuário como um painel interativo, acessível por meio de um botão no topo da tela. O painel é construído utilizando também a diretiva estrutural **ngIf*. O código HTML na figura 6.8 mostra que o painel de leitura só é renderizado quando a variável mostrarLeitura está com valor *true*, o que é alterado pelo clique do usuário no botão com o ícone de livros.

Dentro desse painel, há uma nova verificação condicional, também com **ngIf*, para exibir uma mensagem personalizada caso o *array* listaDeLeitura[ ] esteja vazio. baixo da lista, um botão adicional permite ao usuário executar uma ação coletiva: o "Baixar Todos", este botão chama a função baixarTodos( ), que simula o download dos livros selecionados.

Figura 6.8 - Estrutura da lista de leitura

```
<div *ngIf="mostrarLeitura" class="painel-leitura">
    <h3>Lista de Leitura</h3>
    <div *ngIf="listaDeLeitura.length === 0">Nenhum livro selecionado ainda.</div>
    <mat-list *ngIf="listaDeLeitura.length > 0">
      <mat-list-item *ngFor="let livro of listaDeLeitura">
        {{ livro.titulo }}
        <button mat-icon-button color="warn" (click)="removerDaLeitura(livro)">
          <mat-icon>delete</mat-icon>
        </button>
      </mat-list-item>
      <button mat-raised-button color="primary" (click)="baixarTodos()">Baixar Todos</button>
    </mat-list>
  </div>
```

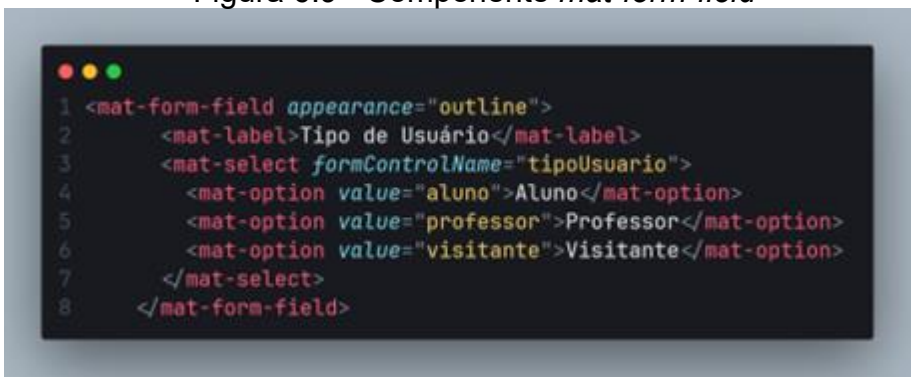
Fonte: Elaborada pela autora, 2025

6.5 COMPONENTES DE SELEÇÃO E ÍCONES

Além dos campos tradicionais de entrada de texto, a aplicação também utiliza o componente *mat-select* dentro de um *mat-form-field* para permitir a seleção de opções predefinidas, como o tipo de usuário. Na figura 6.9, por exemplo, é apresentado o campo de seleção com opções para “Aluno”, “Professor” e “Visitante”, representando diferentes perfis de acesso à biblioteca.

Na figura 6.9 é possível identificar o *mat-select*, componente ideal para menus suspensos e funciona em conjunto com formulários reativos por meio da diretiva *formControlName*. Já o *mat-form-field*, com a propriedade *appearance*="*outline*", confere ao campo um visual mais moderno, com bordas visíveis e rótulo flutuante (*mat-label*).

Figura 6.9 - Componente *mat-form-field*



```
<mat-form-field appearance="outline">
      <mat-label>Tipo de Usuário</mat-label>
      <mat-select formControlName="tipoUsuario">
        <mat-option value="aluno">Aluno</mat-option>
        <mat-option value="professor">Professor</mat-option>
        <mat-option value="visitante">Visitante</mat-option>
      </mat-select>
    </mat-form-field>
```

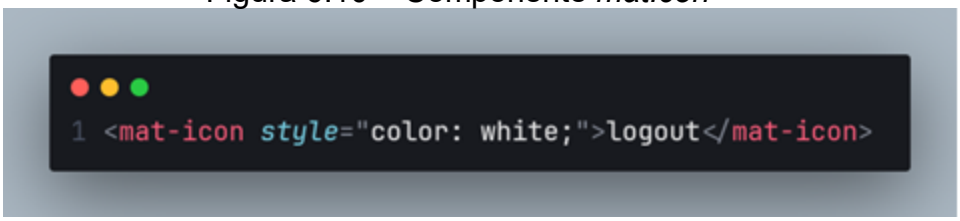
Fonte: Elaborada pela autora, 2025

Outro componente bastante utilizado na aplicação é o *MatIcon*, que faz parte da biblioteca Angular Material e permite incorporar ícones vetoriais seguindo o padrão do Google Material Icons. Esse componente é amplamente aplicado em botões de

ação, tendo em vista que a inclusão de ícones melhora significativamente a experiência do usuário, pois torna as ações mais reconhecíveis visualmente e reduz a necessidade de instruções textuais.

Como mostrado na figura 6.10, o botão de sair da aplicação utiliza o ícone logout dentro de um *mat-icon*, estilizado com *color: white* para se adequar ao contraste visual da interface. Este detalhe demonstra a flexibilidade do *MatIcon*, que pode ser facilmente estilizado com CSS *inline* ou por meio de classes.

Figura 6.10 – Componente *matIcon*



Fonte: Elaborada pela autora, 2025

Essa estrutura demonstra como o Angular permite o controle completo da renderização de elementos com base no estado da aplicação, além de oferecer recursos visuais avançados por meio do Angular Material. Toda a lógica de exibição é mantida limpa, sem manipulação direta do DOM, o que reforça a abstração e a reatividade do *framework*.

O desenvolvimento da Biblioteca Online permitiu explorar, na prática, diversos recursos avançados do framework Angular, aplicados em conjunto com a biblioteca Angular Material para a construção de uma interface moderna e responsiva. Por meio do uso de diretivas estruturais, serviços, formulários reativos e manipulação dinâmica, foi possível criar um sistema funcional e bem estruturado.

Além disso, o projeto reforçou a importância da separação entre lógica e apresentação, bem como da reutilização de componentes, conceitos fundamentais em aplicações modernas baseadas em Single Page Applications (SPA). A experiência proporcionada pela disciplina contribuiu significativamente para a consolidação de conhecimentos em desenvolvimento *frontend* com tecnologias atuais e amplamente utilizadas no mercado.

7 DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS II

Este capítulo apresenta o desenvolvimento de um aplicativo móvel do tipo quiz, elaborado como atividade prática da disciplina de Desenvolvimento para Dispositivos Móveis II juntamente ao Prof. Dr. Thiago Seti Patrício, com o objetivo de aplicar os conceitos de *Intents*, navegação entre telas e interação com o usuário por meio de eventos e elementos multimídia. O projeto foi desenvolvido na linguagem Java, utilizando o Android Studio como ambiente de desenvolvimento integrado, permitindo a criação de uma interface intuitiva e interativa voltada ao sistema operacional Android.

O aplicativo consiste em uma tela inicial que exibe cinco imagens, cada uma representando um tema dentro do universo escolhido para o jogo. Ao selecionar uma das imagens, o usuário é direcionado a uma nova tela, onde é iniciado um *quiz* composto por cinco perguntas de múltipla escolha, com apenas uma alternativa correta em cada questão. O sistema foi projetado para emitir um *feedback* sonoro imediato, utilizando o componente *MediaPlayer*: um som específico para respostas corretas e outro para respostas incorretas.

A proposta principal deste projeto foi demonstrar, de forma prática, a implementação de *Intents* explícitas para navegação entre *Activities*, bem como a manipulação de eventos de clique, controle de fluxo e uso dos recursos de multimídia locais. As *Activities* representam o ponto de interação entre o usuário e o aplicativo, sendo responsáveis por controlar o ciclo de vida da interface e reagir a eventos de entrada *(Android Developers, 2024)*.

Assim, o aplicativo do tipo *quiz* representa um exercício prático essencial para consolidar os conhecimentos adquiridos na disciplina, abrangendo desde o planejamento da interface até a lógica de programação que rege o funcionamento do sistema.

7.1 FUNDAMENTOS E TEORIA

As aplicações Android são estruturadas em *Activities*, que representam telas ou interfaces independentes dentro de um mesmo aplicativo. Cada *Activity* é responsável por controlar a interação entre o usuário e os componentes visuais, além de gerenciar os eventos e ações disparadas por esses elementos. Para viabilizar a

comunicação entre diferentes telas, o Android utiliza o conceito de *Intents*, que são mensagens responsáveis por iniciar *Activities*, transmitir dados e permitir a navegação entre diferentes partes da aplicação (MCCARTHY, 2019).

Outro conceito essencial empregado neste projeto é o uso de eventos de clique, que permitem a captura e o tratamento das ações executadas pelo usuário sobre elementos da interface. Em conjunto com o *listener* setOnClickListener(), é possível associar comportamentos específicos a cada componente visual, garantindo que a aplicação responda dinamicamente às interações.

No contexto do aplicativo desenvolvido, foi também implementado o componente *MediaPlayer*, utilizado para o controle e reprodução de arquivos de áudio. De acordo com a documentação oficial do *Android Developers* (2024), o *MediaPlayer* fornece suporte nativo para a execução de sons e músicas armazenados localmente, possibilitando a criação de *feedbacks* sonoros para reforçar a resposta do usuário durante a execução do *quiz*.

A linguagem de programação adotada foi o Java, amplamente utilizada no ecossistema Android e caracterizada por sua orientação a objetos, portabilidade e robustez, sendo uma linguagem consolidada para o desenvolvimento de sistemas distribuídos e aplicativos que exigem alto nível de controle sobre os componentes da interface e da lógica de execução, sendo por isso apropriada para o desenvolvimento de aplicações móveis.

O desenvolvimento do aplicativo foi realizado utilizando o Android Studio, com a linguagem Java e suporte às APIs mais recentes do ecossistema Android, adotando *compileSdk* 36, *targetSdk* 36 e compatibilidade mínima definida em *minSdk* 34. A arquitetura do projeto foi estruturada em múltiplas telas interativas, respeitando o princípio de separação de responsabilidades, onde cada tela possui função específica no fluxo da aplicação.

A lógica do aplicativo é composta por três telas principais:

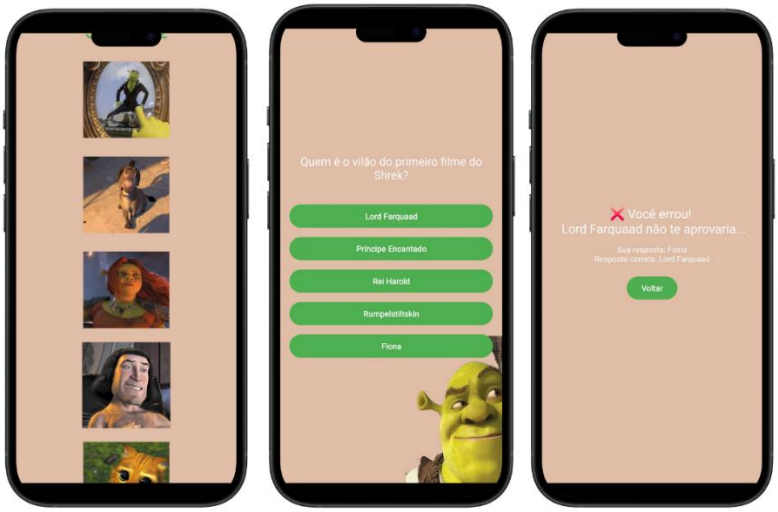
- MainActivity: tela inicial, responsável pela exibição dos personagens e captura do evento de clique do usuário.
- QuizActivity: tela intermediária responsável por apresentar a pergunta correspondente ao personagem escolhido, bem como as cinco alternativas a serem respondidas.

- ResultadoActivity: tela final que apresenta o resultado referente ao acerto ou erro do usuário, executando também a reprodução do som apropriado ao *feedback* interativo.

Além das *Activities*, foram utilizadas classes auxiliares para organização dos dados e constantes, como a classe *Question*, que encapsula os atributos de cada pergunta, e a classe *Constants*, responsável por centralizar chaves de comunicação entre telas e as perguntas cadastradas localmente.

A tela inicial corresponde ao ponto de entrada da aplicação e tem como finalidade direcionar o usuário para a próxima etapa do fluxo por meio da navegação para outra *Activity* via *Intent*. Na Figura 7.1 será ilustrada a tela inicial do sistema executado em ambiente real no dispositivo Android.

Figura 7.1 - Visão geral da aplicação



Fonte: Elaborada pela autora, 2025

7.2 PASTAS E BUILD GRADE

Dentro do projeto, os arquivos e diretórios foram organizados automaticamente pela estrutura padrão da *Integrated Development Environment* (IDE), divididos em pastas específicas para código-fonte, recursos visuais e configurações de compilação.

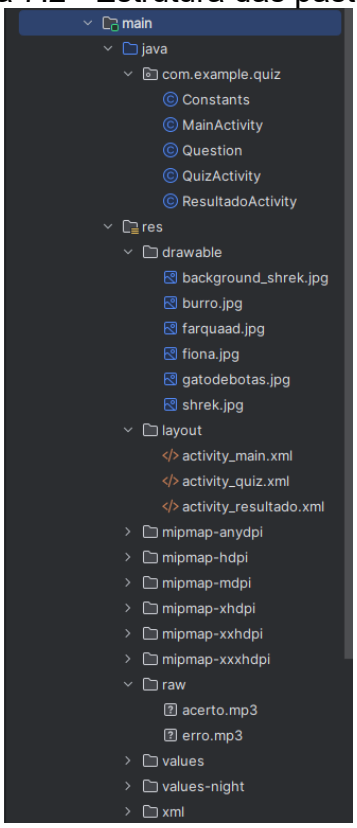
A pasta *java* contém os arquivos de código-fonte em Java, responsáveis pela lógica de funcionamento do aplicativo. Nela, encontram-se as classes que controlam as atividades da aplicação, como a exibição das perguntas, a verificação das respostas e a contagem de acertos do usuário.

Já a pasta *res*/ (*resources*) reúne os arquivos relacionados à interface gráfica, como os layouts em *Extensible Markup Language* (XML), imagens e ícones. Dentro dela, a subpasta layout/ armazena os arquivos XML que definem visualmente as telas do aplicativo, especificando elementos como botões, textos e cores.

Outras subpastas, como *drawable*/, contêm os recursos visuais utilizados nas telas, enquanto a pasta *values*/ centraliza definições de cores e textos reutilizados no projeto, *res*/*raw*/ destinada a recursos brutos, como áudios e vídeos, neste projeto contém dois arquivos de som: acerto.mp3 para o som reproduzido quando o usuário acerta uma questão e erro.mp3 que reproduz o som quando o usuário erra uma questão, esses sons são carregados e executados na classe ResultadoActivity utilizando a classe *MediaPlayer*.

A figura 7.2 demonstra esta estrutura modular, que facilita a manutenção do código e a separação entre a lógica de funcionamento e os elementos visuais da aplicação, permitindo um desenvolvimento mais organizado e eficiente.

Figura 7.2 - Estrutura das pastas

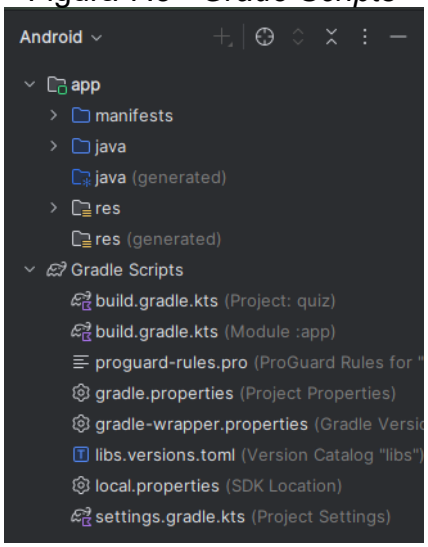


Fonte: Elaborada pela autora, 2025

Além dessas pastas, o Android Studio também gera a pasta *Gradle Scripts*, onde estão os arquivos de configuração responsáveis pela compilação do projeto e pelo gerenciamento de dependências. O arquivo *build.gradle* (nível de módulo) é especialmente importante, pois define as versões do SDK utilizadas, o *namespace* da aplicação, as bibliotecas importadas e o comportamento dos diferentes modos de compilação (debug e release).

O *Gradle* é a ferramenta de automação de compilação usada no Android Studio para gerenciar dependências, configurações de build e empacotamento do aplicativo *(Google Developers, 2024).*

Figura 7.3 - *Grade Scripts*



Fonte: Elaborada pela autora, 2025

No caso deste projeto, o arquivo define, por exemplo, a compatibilidade do código com a versão 11 do Java e a utilização das bibliotecas do *AndroidX*, Material Design e *ConstraintLayout* fundamentais para a construção da interface e a manutenção de boas práticas de usabilidade no aplicativo.

7.3 ACTIVITY PRINCIPAL (MAIN ACTICITY)

A classe MainActivity é responsável por exibir a tela inicial do aplicativo e permitir que o usuário selecione qual personagem deseja responder no quiz. Essa tela representa o ponto de entrada da aplicação, sendo carregada automaticamente ao iniciar o app.

Conforme ilustrado na figura 7.4 é possível identificar que a implementação segue o padrão do ciclo de vida das Activities no Android. Assim que a aplicação é iniciada, o método *onCreate*() é chamado, onde ocorre o carregamento do layout principal por meio do método *setContentView*(), nesse momento, todos os componentes visuais definidos no XML são vinculados às variáveis Java por meio do método *findViewById*().

Figura 7.4 - Método *OnCreate*

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    imgShrek = findViewById(R.id.imgShrek);
    imgFiona = findViewById(R.id.imgFiona);
    imgBurro = findViewById(R.id.imgBurro);
    imgGato = findViewById(R.id.imgGato);
    imgLord = findViewById(R.id.imgFarquaad);

    imgShrek.setOnClickListener( View v -> abrirQuiz( personagem: "shrek"));
    imgFiona.setOnClickListener( View v -> abrirQuiz( personagem: "fiona"));
    imgBurro.setOnClickListener( View v -> abrirQuiz( personagem: "burro"));
    imgGato.setOnClickListener( View v -> abrirQuiz( personagem: "gato"));
    imgLord.setOnClickListener( View v -> abrirQuiz( personagem: "lord"));
}
```

Fonte: Elaborada pela autora, 2025

Em seguida na figura 7.5, cada personagem do universo escolhido para o app é representado por um *ImageView* (como Shrek, Fiona, Burro, Gato de Botas e Lord Farquaad), cada imagem possui um *listener* de clique, responsável por iniciar uma nova *Activity* (QuizActivity) correspondente à pergunta associada àquele personagem.

Figura 7.5 - Método AbrirQuiz

```
@Override
protected void onCreate(Bundle savedInstanceState) {...}

5 usages
private void abrirQuiz(String personagem) {
    Question q = Constants.getQuestionsMap().get(personagem);
    Intent intent = new Intent( packageContext: MainActivity.this, QuizActivity.class);
    intent.putExtra( name: "id", q.getId());
    intent.putExtra( name: "pergunta", q.getQuestion());
    intent.putExtra( name: "imagem", q.getImage());
    intent.putExtra( name: "op1", q.getOptionOne());
    intent.putExtra( name: "op2", q.getOptionTwo());
    intent.putExtra( name: "op3", q.getOptionThree());
    intent.putExtra( name: "op4", q.getOptionFour());
    intent.putExtra( name: "op5", q.getOptionFive());
    intent.putExtra( name: "correta", q.getCorrectAnswer());
    startActivity(intent);
}
}
```

Fonte: Elaborada pela autora, 2025

Quando o usuário seleciona um personagem, o método abrirQuiz() é acionado. Esse método consulta um mapa de perguntas armazenado na classe *Constants*, obtendo o objeto *Question* correspondente ao personagem escolhido. Em seguida, os dados da questão como o texto da pergunta, imagem, opções de resposta e índice da alternativa correta são adicionados ao objeto *Intent* e enviados para a QuizActivity.

Essa estratégia dispensa o uso de banco de dados, simplificando o fluxo do aplicativo e tornando-o mais eficiente para fins didáticos.

7.4 CLASSE CONSTANTS

A classe *Constants* tem a função de centralizar informações fixas e estruturadas utilizadas em diferentes partes do aplicativo, garantindo melhor organização e reutilização do código. Ela segue o princípio de encapsulamento de constantes, evitando a repetição de valores literais em múltiplas classes e facilitando a manutenção futura do sistema.

Logo no início, são definidas três variáveis *public static final String* (EXTRA_ACERTOU, EXTRA_ESCOLHA e EXTRA_CORRETA_TEXTO). Esses identificadores são usados como chaves de comunicação entre *Activities*, permitindo o envio e a recuperação de dados por meio de objetos *Intent*. Por exemplo, ao término de uma pergunta, a *QuizActivity* pode enviar o resultado (se o usuário acertou, qual alternativa escolheu e qual era a correta) para a ResultadoActivity utilizando essas constantes como referência.

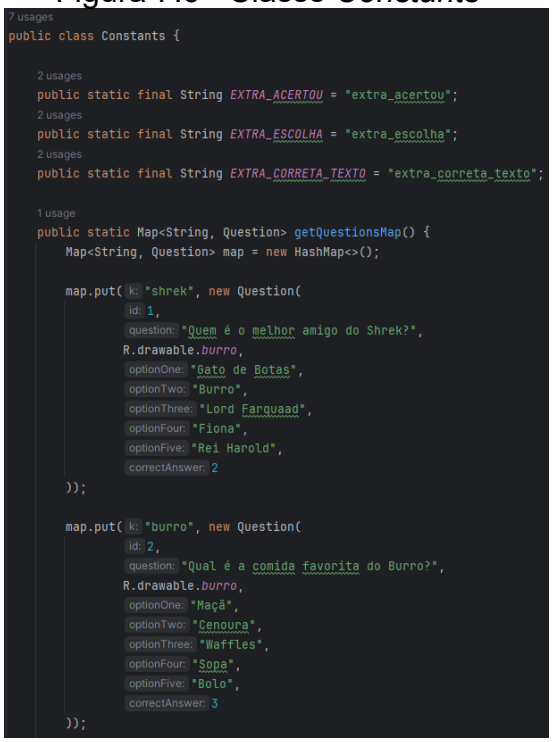
O método estático getQuestionsMap() é o principal componente dessa classe. Ele retorna uma estrutura de dados do tipo Map<String, Question>, onde cada chave representa um personagem do universo Shrek (como *"shrek"*, *"fiona"*, *"burro"*, *"lord"*, *"gato"*) e o valor associado é um objeto da classe *Question*.

Cada instância de *Question* contém os dados necessários para exibir uma pergunta do quiz, incluindo: ID da pergunta, Texto da questão, Imagem ilustrativa (*R.drawable*), Cinco alternativas de resposta, Número da alternativa correta.

Essa estrutura de mapa permite que o aplicativo acesse rapidamente a pergunta correspondente ao personagem escolhido na tela inicial (*MainActivity*). Assim, ao clicar em uma imagem, o app recupera o objeto *Question* correto com base na chave passada e inicia a *QuizActivity* com as informações adequadas. A figura 7.6

mostra uma parte do código do mapa de perguntas armazenadas na classe *Constants*.

Figura 7.6 - Classe *Constants*



```
7 usages
public class Constants {

    2 usages
    public static final String EXTRA_ACERTOU = "extra_acertou";
    2 usages
    public static final String EXTRA_ESCOLHA = "extra_escolha";
    2 usages
    public static final String EXTRA_CORRETA_TEXTO = "extra_correta_texto";

    1 usage
    public static Map<String, Question> getQuestionsMap() {
        Map<String, Question> map = new HashMap<>();

        map.put( k: "shrek", new Question(
                id: 1,
                question: "Quem é o melhor amigo do Shrek?",
                R.drawable.burro,
                optionOne: "Gato de Botas",
                optionTwo: "Burro",
                optionThree: "Lord Farquaad",
                optionFour: "Fiona",
                optionFive: "Rei Harold",
                correctAnswer: 2
        ));

        map.put( k: "burro", new Question(
                id: 2,
                question: "Qual é a comida favorita do Burro?",
                R.drawable.burro,
                optionOne: "Maçã",
                optionTwo: "Cenoura",
                optionThree: "Waffles",
                optionFour: "Sopa",
                optionFive: "Bolo",
                correctAnswer: 3
        ));
```

Fonte: Elaborada pela autora, 2025

7.5 CLASSE QUESTION

A classe *Question* é responsável por representar a estrutura de cada pergunta do quiz, funcionando como o modelo de dados central da aplicação. Cada instância dessa classe contém todas as informações necessárias para exibir e validar uma questão durante a execução do aplicativo.

Sua implementação segue os princípios da programação orientada a objetos (POO)**,** mais especificamente os conceitos de abstração e encapsulamento. Por meio da abstração, a classe transforma o conceito de “pergunta” em um objeto do mundo real, com propriedades e comportamentos definidos. Já o encapsulamento garante que os atributos sejam acessados apenas pelos métodos apropriados, mantendo a integridade dos dados.

A classe é composta pelos seguintes atributos:

- id (int): Identificador numérico único da pergunta;
- question (*String*)**:** Texto da pergunta exibida ao usuário;

- image (int): Referência à imagem associada à questão (armazenada no diretório res/drawable);
- optionOne a optionFive (*String*): As cinco alternativas de resposta disponíveis;
- correctAnswer (int): Índice que indica qual das alternativas é a correta.

O construtor da classe recebe todos esses parâmetros e os atribui às variáveis de instância, permitindo criar objetos *Question* completos e prontos para uso no aplicativo. Além disso, a classe disponibiliza métodos *getters* para acesso controlado aos dados, garantindo que outras classes possam recuperar as informações sem modificar o conteúdo diretamente.

Por meio da figura 7.7 é possível ver que essa estrutura favorece a modularidade do código, pois separa a camada de dados da lógica de funcionamento. Assim, quando a classe *Constants* cria o mapa de perguntas, ela simplesmente instancia objetos *Question*, tornando o código mais organizado e fácil de manter.

Figura 7.7 - Classe *Question*

```
5 usages
public Question(int id, String question, int image,
                String optionOne, String optionTwo, String optionThree,
                String optionFour, String optionFive, int correctAnswer) {
    this.id = id;
    this.question = question;
    this.image = image;
    this.optionOne = optionOne;
    this.optionTwo = optionTwo;
    this.optionThree = optionThree;
    this.optionFour = optionFour;
    this.optionFive = optionFive;
    this.correctAnswer = correctAnswer;
}

1 usage
public int getId() { return id; }
1 usage
public String getQuestion() { return question; }
1 usage
public int getImage() { return image; }
1 usage
public String getOptionOne() { return optionOne; }
1 usage
public String getOptionTwo() { return optionTwo; }
1 usage
public String getOptionThree() { return optionThree; }
1 usage
public String getOptionFour() { return optionFour; }
1 usage
public String getOptionFive() { return optionFive; }
1 usage
public int getCorrectAnswer() { return correctAnswer; }
}
```

Fonte: Elaborada pela autora, 2025

7.6 CLASSE QUIZ ACTIVITY

A classe *QuizActivity* representa a tela principal de execução do quiz, sendo responsável por exibir a pergunta selecionada, suas alternativas e processar as respostas do usuário. Essa classe é instanciada a partir da *MainActivity*, por meio de uma *Intent*, que envia os dados da pergunta (texto, imagem, alternativas e resposta correta) para esta nova tela, visível na figura 7.8

Figura 7.8 – *Intent* responsável pelo envio de dados

```
Intent intent = getIntent();
txtPergunta.setText(intent.getStringExtra( name: "pergunta"));
imgPersonagem.setImageResource(intent.getIntExtra( name: "imagem", R.drawable.shrek));
btnOpcao1.setText(intent.getStringExtra( name: "op1"));
btnOpcao2.setText(intent.getStringExtra( name: "op2"));
btnOpcao3.setText(intent.getStringExtra( name: "op3"));
btnOpcao4.setText(intent.getStringExtra( name: "op4"));
btnOpcao5.setText(intent.getStringExtra( name: "op5"));
respostaCorreta = intent.getIntExtra( name: "correta",  defaultValue: 1);
```

Fonte: Elaborada pela autora, 2025

Ao ser criada, a *QuizActivity* recupera as informações enviadas através dos métodos *getIntent*() e *getExtras*(), garantindo que a questão correspondente ao personagem selecionado seja carregada corretamente. A partir disso, o layout da atividade é populado dinamicamente com os elementos visuais, como o texto da pergunta, a imagem e os botões de múltipla escolha.

Figura 7.9 - Métodos *getIntent* e *getExtras*

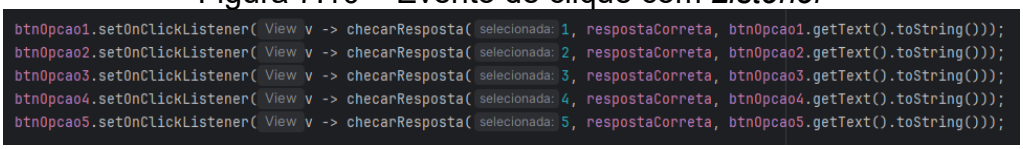
```
txtPergunta = findViewById(R.id.txtPergunta);
imgPersonagem = findViewById(R.id.imgPersonagem);
btnOpcao1 = findViewById(R.id.btnOpcao1);
btnOpcao2 = findViewById(R.id.btnOpcao2);
btnOpcao3 = findViewById(R.id.btnOpcao3);
btnOpcao4 = findViewById(R.id.btnOpcao4);
btnOpcao5 = findViewById(R.id.btnOpcao5);
```

Fonte: Elaborada pela autora, 2025

Cada botão de resposta é associado a um *listener* que identifica qual opção o usuário selecionou, após a escolha, o sistema realiza uma verificação comparando o índice da opção escolhida com o índice da resposta correta, previamente armazenada. Se o usuário acertar, é reproduzido um som de acerto, indicando

feedback positivo, se o usuário errar, é reproduzido um som de erro, sinalizando o resultado incorreto.

Figura 7.10 – Evento de clique com *Listener*



Fonte: Elaborada pela autora, 2025

Esses efeitos sonoros são armazenados no diretório *res/raw* e são executados por meio da classe *MediaPlayer*, que possibilita a reprodução de arquivos de áudio no Android, esse recurso contribui para a interatividade e imersão da aplicação.

Após o usuário responder à questão, a QuizActivity utiliza uma nova *Intent* para navegar até a ResultadoActivity, enviando junto as informações necessárias para exibir o resultado final. Esse fluxo demonstra o uso do conceito de *Intents* explícitas, que permitem a comunicação e transição de dados entres diferentes telas de aplicação, um dos objetivos principais da atividade prática proposta pelo professor.

7.7 CLASSE RESULTADO ACTIVITY

A classe ResultadoActivity é responsável por exibir o resultado final do quiz, informando ao usuário se a resposta escolhida estava correta ou incorreta, detalhando qual alternativa foi selecionada e qual era a resposta correta. Esta tela representa o fechamento do ciclo de interação iniciado na MainActivity e continuado na QuizActivity. Quando a atividade é criada, o método *onCreate*() é chamado, iniciando o carregamento do layout correspondente por meio de *setContentView*(), os componentes visuais, como *TextView* e *Button*, são vinculados às variáveis Java utilizando *findViewById*().

A lógica é implementada por meio de uma estrutura condicional if-else que indica se a resposta do usuário corresponde a resposta correta, a ResultadoActivity recebe os dados da resposta através de uma *Intent***,** e a variável acertou é do tipo *boolean, ou seja,* é um tipo de dado que só pode assumir dois valores possíveis: *true* ou *false*, obtida a partir da *Intent* enviada pela QuizActivity, utilizando as constantes definidas na classe *Constants*. Esses dados incluem:

- EXTRA_ACERTOU: Indica se o usuário acertou ou errou a questão;
- EXTRA_ESCOLHA: Registra o texto da alternativa escolhida pelo usuário;
- EXTRA_CORRETA_TEXTO: Contém o texto da alternativa correta.

Com base no valor de EXTRA_ACERTOU, a aplicação determina a mensagem de resultado exibida ao usuário e seleciona o feedback sonoro apropriado, arquivos de áudio armazenados no diretório *res/raw* são reproduzidos utilizando a classe *MediaPlayer*, proporcionando uma experiência interativa. Além disso, a tela exibe detalhes adicionais, informando ao usuário qual alternativa ele escolheu e qual era a resposta correta, e o botão “Jogar Novamente” permite retornar à *MainActivity*, reiniciando o *quiz* para uma nova tentativa. A Figura 7.11 demonstra o trecho do código que processa o resultado e reproduz o som.

Figura 7.11 - Uso do *MediaPlayer*

```
boolean acertou = getIntent().getBooleanExtra(Constants.EXTRA_ACERTOU, defaultValue: false);
String escolha = getIntent().getStringExtra(Constants.EXTRA_ESCOLHA);
String correta = getIntent().getStringExtra(Constants.EXTRA_CORRETA_TEXTO);

if (acertou) {
    txtResultado.setText("🎉 Você acertou! \n Shrek ficaria orgulhoso!");
    mediaPlayer = MediaPlayer.create( context: this, R.raw.acerto); // toca som de acerto
} else {
    txtResultado.setText("❌ Você errou! \n Lord Farquaad não te aprovaria...");
    mediaPlayer = MediaPlayer.create( context: this, R.raw.erro); // toca som de erro
}

txtDetalhe.setText("Sua resposta: " + (escolha != null ? escolha : " - ") +
        "\nResposta correta: " + (correta != null ? correta : " - "));

if (mediaPlayer != null) {
    mediaPlayer.start();
}

btnJogarNovamente.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { finish(); // volta para a main }
});
}
```

Fonte: Elaborada pela autora, 2025

Esse é o ponto central da lógica de decisão e *feedback* na aplicação, a classe ResultadoActivity é responsável por apresentar o desempenho final do usuário no quiz. Nela, os dados enviados pela atividade anterior são processados e exibidos na tela por meio de elementos gráficos definidos no XML concluindo a interação do usuário com o aplicativo, fornecendo um retorno visual e lógico sobre seu resultado.

7.8 ESTRUTURAS DE LAYOUTS EM XML

No desenvolvimento de aplicativos Android, os arquivos XML desempenham um papel fundamental na definição da interface gráfica do usuário (UI), são

responsáveis por descrever visualmente como cada tela será apresentada, organizando elementos como botões, textos, imagens, campos interativos e componentes estruturais.

No Android Studio, cada tela do aplicativo costuma ter um arquivo XML associado, localizado dentro da pasta *res/layout*. Esses arquivos funcionam como um "molde" da interface, permitindo que a camada visual seja separada da lógica do aplicativo, seguindo princípios de modularidade e boas práticas de desenvolvimento.

No contexto deste projeto, foram utilizados três arquivos XML principais, cada um representando uma tela distinta:

- activity_main.xml: Responsável pela tela inicial com os personagens;
- activity_quiz.xml: Onde as perguntas e alternativas são exibidas;
- activity_resultado.xml: Destinado à apresentação do resultado final do quiz.

7.9 ACTIVITY MAIN XML

O arquivo activity_main.xml é responsável por definir a interface visual da tela inicial do aplicativo. Nesta tela, o usuário pode escolher qual personagem deseja utilizar para iniciar o quiz, sendo um ponto central da navegação do sistema. O layout foi estruturado utilizando uma combinação de elementos fundamentais da construção de interfaces no Android: *ScrollView*, *LinearLayout*, componentes de texto e múltiplos *ImageViews*.

O componente raiz do layout é um *ScrollView* apresentado na figura 7.12, escolhido para garantir que o usuário consiga visualizar todos os elementos, independentemente do tamanho da tela do dispositivo.

Figura 7.12 - Componente ScrollView

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#E2BEA8">
```

Fonte: Elaborada pela autora, 2025

Dentro dele, encontra-se um *LinearLayout* com orientação vertical, responsável por organizar os elementos de forma sequencial de cima para baixo, mantendo a estrutura simples e intuitiva.

Figura 7.13 - Componente *LinearLayout*

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="vertical"
    android:padding="16dp">
```

Fonte: Elaborada pela autora, 2025

Na parte superior da tela, um *TextView* exibe o título *"Shrek Quiz"*, com destaque visual obtido por meio do uso de texto em negrito, cor personalizada e tamanho ampliado.

Figura 7.14 - Componente TextView

```
<TextView
    android:id="@+id/tvTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:paddingTop="60dp"
    android:paddingBottom="24dp"
    android:text="Shrek Quiz"
    android:textColor="@color/verde"
    android:textStyle="bold"
    android:textSize="30sp" />
```

Fonte: Elaborada pela autora, 2025

Em seguida, são incluídos cinco *ImageViews*, cada um representando um personagem: Shrek, Burro, Fiona, Lord Farquaad e Gato de Botas, cada imagem é exibida com dimensões uniformes, margens adequadas e o atributo *adjustViewBounds*="*true*", que permite que o Android ajuste o tamanho da imagem proporcionalmente, evitando distorções, a figura 7.15 apresenta a exibição da imagem de um dos personagens, o padrão se repete para todos os outros.

Figura 7.15 - Componente *ImageView*

```
<ImageView
    android:id="@+id/imgFarquaad"
    android:layout_width="170dp"
    android:layout_height="150dp"
    android:layout_gravity="center"
    android:layout_marginBottom="16dp"
    android:adjustViewBounds="true"
    android:contentDescription="@string/lord_farquaad"
    android:padding="8dp"
    android:scaleType="centerCrop"
    android:src="@drawable/farquaad" />
```

Fonte: Elaborada pela autora, 2025

Além disso, o uso de *contentDescription* em cada *ImageView* demonstra a adoção de boas práticas de acessibilidade, permitindo que tecnologias assistivas identifiquem corretamente cada elemento visual.

7.10 ACTIVITY QUIZ XML

O arquivo activity_quiz.xml define a interface utilizada durante a execução do quiz, apresentando ao usuário a pergunta atual, suas alternativas e, quando aplicável, a imagem correspondente ao personagem da questão, diferente da tela principal, esse layout organiza elementos funcionais diretamente relacionados à mecânica do jogo visível na figura 7.16.

Figura 7.16 - Estrutura do *TextView* da pergunta

```
<TextView
    android:id="@+id/txtPergunta"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="20sp"
    android:paddingBottom="18dp"
    android:gravity="center"
    android:layout_marginBottom="16dp"
    android:paddingTop="32dp"
    android:textColor="@android:color/white"/>
```

Fonte: Elaborada pela autora, 2025

Logo no topo da estrutura está o *TextView* txtPergunta, responsável por exibir dinamicamente o enunciado de cada questão. Seu estilo utiliza fonte maior e alinhamento central para facilitar a leitura durante o avanço do *quiz*.

Em seguida, são definidos cinco botões de alternativa (btnResposta1 a btnResposta5). Esses componentes representam as opções que o usuário pode selecionar, cada botão possui largura total e margens inferiores para garantir espaçamento adequado entre eles, além da aplicação da cor personalizada do projeto, mantendo coerência visual com o restante da aplicação. A lógica de identificação da resposta correta é tratada na *QuizActivity*, que utiliza esses botões como gatilho para a validação das escolhas do usuário, a figura 7.17 mostra apenas a primeira opção, mas todos os botões seguem o mesmo padrão.

Figura 7.17 - Botões de alternativa

```
<Button
    android:id="@+id/btnOpcao1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:backgroundTint="@color/verde"
    android:textColor="@android:color/white"/>
```

Fonte: Elaborada pela autora, 2025

No final do layout encontra-se o *ImageView* imgPersonagem, que tem a função de exibir a imagem do personagem relacionado à pergunta atual. Inicialmente, o componente está configurado com *visibility="gone"* para ser exibido somente quando o enunciado exigir, comportamento controlado pela camada lógica da activity.

Figura 7.18 - Estrutura do *ImageView* do personagem

```
<ImageView
    android:id="@+id/imgPersonagem"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:layout_gravity="center"
    android:padding="16dp"
    android:src="@drawable/shrek"
    android:visibility="gone"/>
```

Fonte: Elaborada pela autora, 2025

Assim, o layout desta tela não apenas organiza elementos visuais, mas estabelece a estrutura necessária para o funcionamento interativo do *quiz*.

7.11 ACTIVITY RESULTADO XML

O arquivo *activity_resultado.xml* define a interface responsável por apresentar ao usuário o resultado final após responder a uma pergunta do *quiz*. Essa tela encerra o ciclo iniciado na *QuizActivity*, exibindo informações claras sobre o desempenho do usuário, além de oferecer a opção de retornar à tela inicial para continuar a utilização do aplicativo.

Figura 7.19 - Estrutura do TextView do resultado

```
<TextView
    android:id="@+id/txtResultado"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="22sp"
    android:paddingBottom="12dp"
    android:gravity="center"
    android:textColor="@android:color/white"/>
```

Fonte: Elaborada pela autora, 2025

O primeiro elemento do layout é o *TextView* txtResultado, responsável por exibir a informação central da tela, indicando se o usuário acertou ou errou a pergunta. Seu tamanho de fonte maior e o alinhamento central contribuem para uma leitura imediata e objetiva.

Figura 7.20 - TextView secundário (detalhes)

```
<TextView
    android:id="@+id/txtDetalhe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingBottom="18dp"
    android:gravity="center"
    android:textColor="@android:color/white"/>
```

Fonte: Elaborada pela autora, 2025

Após, o *TextView* txtDetalhe apresenta informações complementares, como a alternativa escolhida pelo usuário e qual seria a resposta correta, esse componente torna a experiência mais informativa e educativa.

O layout também inclui um botão (*btnJogarNovamente*), que permite ao usuário retornar à tela inicial, ele utiliza a mesma cor personalizada aplicada aos botões da tela de perguntas, mantendo a coerência visual entre as telas do aplicativo.

Figura 7.21 - Botão para jogar novamente

```
<Button
    android:id="@+id/btnJogarNovamente"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/voltar"
    android:backgroundTint="@color/verde"
    android:textColor="@android:color/white"/>
```

Fonte: Elaborada pela autora, 2025

A Activity Resultado possui uma estrutura simples, porém funcional, focada em transmitir as informações finais do quiz de maneira clara. O uso de *LinearLayout* com orientação vertical, alinhamento central e espaçamento consistente garante que todos os elementos sejam exibidos de forma organizada. Essa tela conclui o fluxo principal da aplicação, reforçando o uso de comunicação entre activities por meio de *Intents*, iniciado anteriormente na *QuizActivity*.

8 MANUAL DO USUÁRIO

A construção do portfólio foi orientada por referências visuais alinhadas ao estilo pessoal, refletidas na escolha das cores, tipografias e elementos gráficos utilizados, a seleção desses componentes buscou transmitir uma identidade coerente, valorizando aspectos estéticos que representam a personalidade de forma sutil e harmoniosa.

Para manter coerência estética e reforçar a personalidade do projeto, foi definida uma paleta composta por três cores principais apresentadas na figura 8.1.

Figura 8.1 - Paleta de cores do portfólio

Fonte: Elaborada pela autora, 2025

As tonalidades escolhidas apresentam contraste equilibrado entre cores suaves e tons mais intensos, resultando em uma composição visual que favorece a leitura, destaca os elementos principais e cria uma atmosfera acolhedora e harmoniosa reforçando a personalidade do projeto.

Embora o portfólio não possua um logo tradicional, foi desenvolvido um elemento característico que funciona como uma espécie de “assinatura” a partir da fonte Brittany, selecionada por seu traço caligráfico fluido e pela estética que remete a personalização e identidade própria.

Figura 8.2 - Assinatura visual

Fonte: Elaborada pela autora, 2025

A seleção das fontes foi pensada para combinar estética pessoal com funcionalidade. Foram escolhidas duas famílias tipográficas, cada uma desempenhando um papel específico, sendo elas Yeseva One e Roboto.

Figura 8.3 - Fontes utilizadas

Fonte: Elaborada pela autora, 2025

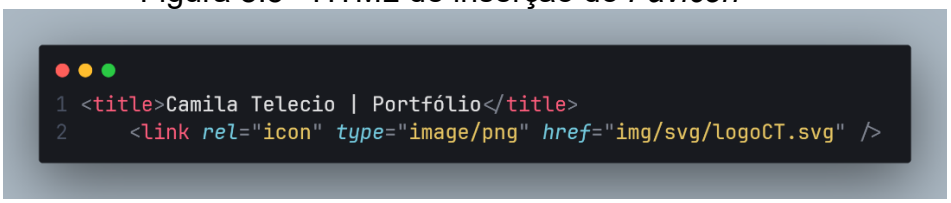
O portfólio conta com um favicon próprio, utilizado para facilitar a identificação da página quando diversas abas do navegador estão abertas. O elemento escolhido foi uma estrela, que faz parte da identidade visual e se tornou um elemento marcante da página.

Figura 8.4 - Favicon do portfólio

Fonte: Elaborada pela autora, 2025

Para incorporá-lo ao projeto, foi utilizado o elemento <link> no cabeçalho do documento HTML, fazendo referência direta ao arquivo do ícone. Essa configuração permite que o navegador reconheça automaticamente o favicon e o associe ao título da página.

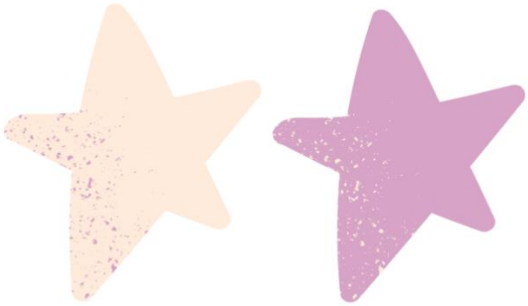
Figura 8.5 - HTML de inserção do *Favicon*



Fonte: Elaborada pela autora, 2025

As composições visuais do portfólio também incorporam outros elementos gráficos que reforçam a identidade do projeto, entre eles, destacam-se as estrelas estilizadas, representadas por formas orgânicas com textura. Esses elementos utilizam tonalidades que se harmonizam com a paleta definida, alternando entre cores claras e médias para criar variação visual.

Figura 8.6 - Elementos de estrelas



Fonte: Elaborada pela autora, 2025

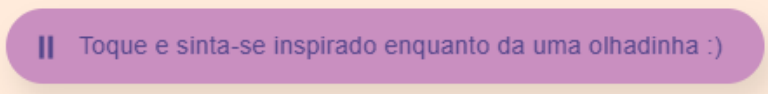
Além disso, o portfólio utiliza uma ilustração de uma personagem estilizada, escolhida por refletir de forma simbólica, características associadas à autora. A figura serve como um elemento representativo dentro da identidade visual, reforçando o caráter pessoal do projeto.

Figura 8.7 - Personagem ilustrada

Fonte: Elaborada pela autora, 2025

Além das páginas tradicionais que apresentam os projetos, experiências e informações profissionais, o portfólio conta com duas funcionalidades interativas, um deles é pequeno tocador de música integrado, localizado na parte inferior da página, desenvolvido com *JavaScript*, ele permite ao usuário interagir com a trilha sonora selecionada para acompanhar a navegação.

Figura 8.8 - *Media Player*



Fonte: Elaborada pela autora, 2025

O portfólio inclui um *easter egg* disponível aos usuários que acessarem via *desktop*, que também foi desenvolvido com *JavaScript*, uma funcionalidade oculta que pode ser descoberta pelo usuário durante a navegação, e ao ser ativada revela um conteúdo especial desenvolvido exclusivamente para quem a encontra.

Figura 8.9 - *Easter Egg*

Fonte: Elaborada pela autora, 2025

O sistema monitora discretamente a digitação no teclado durante a navegação, e a ativação ocorre mediante a digitação da palavra-chave “segredo”, ao ser reconhecida, o script aciona a abertura de um modal exclusivo.

A estrutura do portfólio foi desenvolvida utilizando HTML, CSS, *JavaScript* e o *framework Bootstrap*, que ofereceu suporte à organização do layout e à implementação de recursos responsivos, a combinação dessas tecnologias permitiu a criação de uma interface funcional, leve e adaptada.

A versão *mobile* recebeu atenção especial, considerando as limitações de espaço e a necessidade de priorizar legibilidade e praticidade, o *Bootstrap* forneceu componentes flexíveis que facilitaram o reposicionamento de elementos, a adaptação das tipografias e o ajuste das proporções em telas menores.

Figura 8.10 - *Layout* da tela inicial (versão *mobile*)

Fonte: Elaborada pela autora, 2025

O player de áudio é exibido em um formato reduzido, os cartões de projetos, que aparecem dispostos lado a lado em telas maiores, passam a ser organizados verticalmente no mobile, a ilustração principal da personagem que compõe a tela inicial do portfólio deixa de ser exibida em telas muito pequenas. Essa decisão foi tomada para otimizar o espaço disponível e priorizar elementos de uso direto.

O conjunto de adaptações garante que o portfólio mantenha consistência estética e funcionalidade independentemente do tamanho da tela.

9 CONSIDERAÇÕES FINAIS

A finalização deste portfólio digital permitiu uma reflexão ampla sobre o percurso acadêmico e sobre o desenvolvimento das competências adquiridas ao longo da graduação em Sistemas para Internet, a produção contínua de projetos durante o curso demonstrou-se fundamental para a consolidação do aprendizado, evidenciando como a prática constante contribui para o aprimoramento técnico e criativo.

O portfólio construído reúne os trabalhos desenvolvidos em diferentes disciplinas, apresentando não apenas o resultado final de cada atividade, mas também a evolução das habilidades ao longo do tempo.

A disponibilização em ambiente online facilita o acesso ao conteúdo e amplia seu alcance, tornando-o um recurso útil para potenciais empregadores e demais interessados.

Durante o processo de criação de cada projeto, foi possível compreender de forma mais concreta a dinâmica envolvida no desenvolvimento web, desde a concepção da ideia até sua implementação prática, a resolução de problemas, o estudo de novas ferramentas e a necessidade de tomada de decisões adequadas contribuíram para fortalecer o pensamento crítico e a capacidade de adaptação diante de desafios.

Além dos aspectos técnicos, a vivência acadêmica proporcionou crescimento pessoal, aprimoramento de habilidades interpessoais e maior entendimento da importância da organização, comunicação e colaboração em ambientes de desenvolvimento, esses elementos complementares desempenharam papel significativo na formação profissional.

Assim, este portfólio cumpre o papel de registrar a trajetória percorrida, demonstrar os conhecimentos adquiridos e representar o resultado de um processo consistente de aprendizagem.

Ele se consolida como um instrumento que evidencia competências e prepara o caminho para futuras oportunidades na área profissional.

REFERÊNCIAS

MACHADO, M. F. R. C.; SILVA, F. H.; SAKALAUSKAS, S. R. As contribuições do portfólio digital como instrumento de avaliação. Intersaberes, v. 14, n. 33, p. 130-150, 2019. Disponível em: https://www.researchgate.net/publication/331604601_As_contribuicoes_do_portfolio_digital_como_instrumento_de_avaliacao. Acesso em: 19 set. 2025.

APOSTILA DE BANCO DE DADOS E SQL. Disponível em: https://www.josevalter.com.br/download/sql/Apostila%20de%20Banco%20de%20Dados%20e%20SQL.pdf. Acesso em: 20 set. 2025.

BEAULIEU, Alan. Aprendendo SQL: Dominando os Fundamentos de SQL. Novatec Editora, 2019. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=63iYDwAAQBAJ&oi=fnd&pg=PT8&dq=SQL+banco+de+dados&ots=6MHn2k84gl&sig=lVvaaXKuh9B8MsRpeWnX3d_VjpY&redir_esc=y#v=onepage&q=SQL%20banco%20de%20dados&f=false. Acesso em: 20 set. 2025.

BIERMAN, Gavin; ABADI, Martín; TORGERSEN, Mads. Understanding TypeScript. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP), 28., 2014, Uppsala, Suécia. Anais… Berlim, Heidelberg: Springer, 2014. p. 257–281. Disponível em: https://link.springer.com/chapter/10.1007/978-3-662-44202-9_11. Acesso em: 19 set. 2025.

BRASIL. Ministério da Saúde. DATASUS. Sistema Gerenciador de Banco de Dados (SGBD). Disponível em: https://datasus.saude.gov.br/glossario/sistema-gerenciador-de-banco-de-dados-sgbd/. Acesso em: 18 set. 2025.

FREEMAN, Adam. Pro Angular 9: build powerful and dynamic web apps. New York: Apress, 2020. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=ji3rDwAAQBAJ&oi=fnd&pg=PR5&dq=Freeman+e+Robson++angular&ots=EIbnxk0mxm&sig=tI_pqHWtPJfvf74OnZ4TiVNNCCM&redir_esc=y#v=onepage&q&f=false. Acesso em: 3 out. 2025.

GRILLO, Filipe Del Nero; FORTES, Renata Pontin de Mattos. Aprendendo JavaScript. São Carlos: UFSCar, 2008. Disponível em: http://www2.dc.ufscar.br/~renata/apostilas/js/apostila.pdf. Acesso em: 19 set. 2025.

MACÁRIO, Carla Geovana do N.; BALDO, Stefano Monteiro. O modelo relacional. Unicamp. Disponível em: https://www.ic.unicamp.br/~geovane/mo410-091/Ch03-RM-Resumo.pdf. Acesso em: 18 set. 2025.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. Sistema de Banco de Dados. 3. ed. Makron Books, 2005. Disponível em: https://www.scribd.com/document/554459784/Sistema-de-Banco-de-Dados#page=1. Acesso em: 19 set. 2025.

WARGO, John M. Learning Progressive Web Apps. 1. ed. Boston: Addison-Wesley Professional, 2020. Disponível em: https://books.google.com.br/books?hl=pt-

BR&lr=&id=NX_QDwAAQBAJ&oi=fnd&pg=PT13&dq=Wargo,+2019+angular+material&ots=Z6EwnXt8jy&sig=Pf7IUfyZH7Kskkfs95mbHAR_Zrk&redir_esc=y#v=onepage&q&f=false. Acesso em: 15 set. 2025.