O desenvolvimento de uma ferramenta web para edição de partituras

Carlos Henrique Rodrigues de Moraes

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Monte Carmelo - MG
2025

Carlos Henrique Rodrigues de Moraes

O desenvolvimento de uma ferramenta web para edição de partituras

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: Prof. Dr. Thiago Henrique Pereira Silva

Monte Carmelo - MG
2025

Resumo

A edição de partituras musicais por meios digitais ainda é uma área pouco explorada pelas engenharias e pelo desenvolvimento de software, tanto por se tratar de um tema especializado quanto pelos desafios técnicos envolvidos. A representação da notação musical ocidental moderna pode se mostrar visualmente complexa e, desta forma, as ferramentas disponíveis para a edição via meios digitais tendem a crescer em complexidade. Essa complexidade impõe limitações quanto à plataforma e à usabilidade desses softwares, restringindo seu uso a pessoas que já possuem familiaridade tanto com notação musical quanto com sistemas digitais. Este trabalho desenvolveu um sistema de edição de partituras interativo para a web, implementando ferramentas simples e intuitivas de se utilizar considerando quaisquer usuários, com uma avaliação positiva de usabilidade realizada por ferramentas automáticas.

Palavras-chave: Música, Edição de Partituras, Desenvolvimento Web, Engenharia de Software, Usabilidade.

Lista de ilustrações

Figura 1 – Exemplo com elementos da notação musical ocidental. 6
Figura 2 – Exemplo da interface do software MuseScore. 8
Figura 3 – Diagrama de classes do projeto. 20
Figura 4 – Subdivisões das figuras musicais. Adaptada de Chediak (2009) 21
Figura 5 – Exemplo de um compasso completo na fórmula de 4/4 21
Figura 6 – Representação das notas em cifra. 22
Figura 7 – Exemplo demonstrando a relação entre código e notas na pauta. 22
Figura 8 – Tela inicial do aplicativo. 25
Figura 9 – Menu 'Geral'. 26
Figura 10 – Menu 'Player'. 26
Figura 11 – Menu *Notas* com as opções de entrada *Notas* e *Teclado* 27
Figura 12 – Teclado no modo expandido. 28
Figura 13 – Menu 'Compassos' 28
Figura 14 – Menu 'Tempo' 29
Figura 15 – Menu 'Tonalidade' 29

Sumário

1 INTRODUÇÃO 6
1.1 Objetivos 8
1.1.1 Objetivo Geral 8
1.1.2 Objetivos Específicos 9
1.2 Estrutura do Trabalho 9

2 FUNDAMENTAÇÃO TEÓRICA 10
2.1 Desenvolvimento Ágil 10
2.2 Usabilidade 11
2.3 Desenvolvimento Web 11
2.3.1 SPA — Single Page Application 12
2.3.2 React 12
2.3.3 VexFlow 12
2.3.4 Tone.js 13
2.4 Teoria e Notação Musical 13
2.5 Trabalhos Relacionados 13

3 METODOLOGIA 16
3.1 Ambiente de Desenvolvimento 16
3.1.1 Node.js e Vite 16
3.1.2 Versionamento e CI/CD 17
3.2 Desenvolvimento Web 17
3.2.1 Typescript 17
3.2.2 React 18
3.2.3 Gerenciamento de Estado 18
3.3 Notação Musical 20
3.3.1 Reprodução Sonora 23
3.4 Usabilidade 23

3.4.1 Interface e estilização 23
3.4.2 Exportação em PDF e PNG 24

4 EDITOR DE PARTITURAS 25

5 CONCLUSÃO 30
5.1 Principais Contribuições 31
5.2 Trabalhos Futuros 31

REFERÊNCIAS 33

Capítulo 1

Introdução

A representação da notação musical ocidental moderna em meios digitais apresenta uma série de desafios peculiares, principalmente devido às suas diferenças em relação às línguas escritas. Devemos considerar também que computadores pessoais foram projetados inicialmente para operações por meio de linha de comando, o que os tornou inadequados para lidar com linguagens visuais e simbólicas como a notação musical (SWAINE; FREIBERGER, 2014).

Nas partituras, as notas são determinadas pela altura em que a figura musical está posicionada na pauta, e o ritmo é representado pela sequência de notas ao longo das linhas, onde os diferentes tipos de figura musical representam as durações de cada nota ou silêncio na música. Além destas características citadas, podemos observar na Figura 1 diversos outros elementos visuais que fazem parte da notação básica.

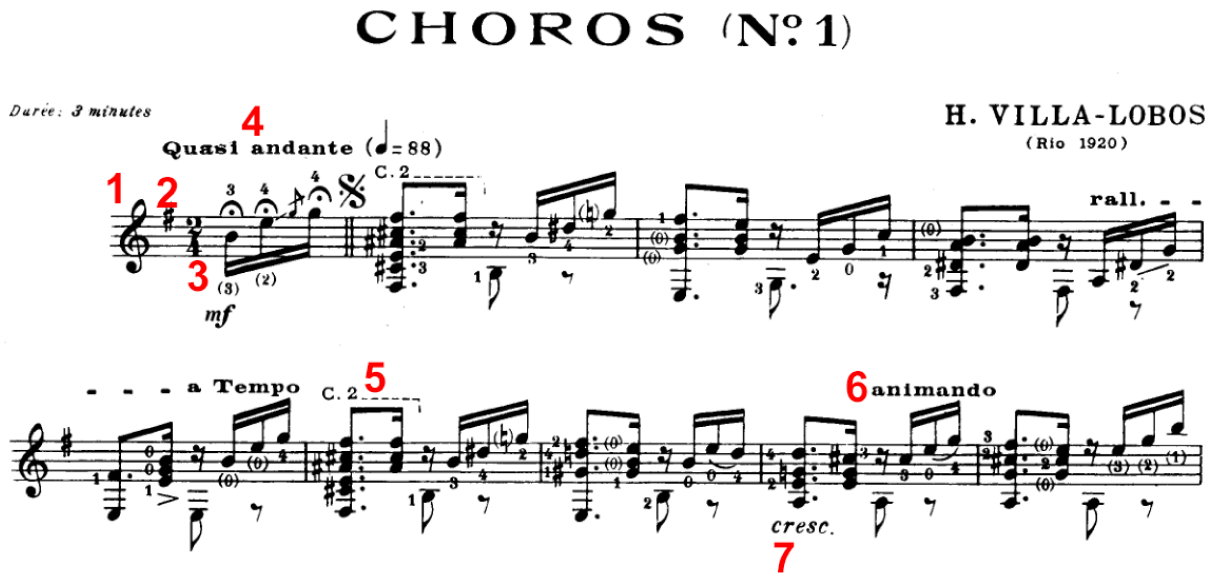

Figura 1 – Exemplo com elementos da notação musical ocidental.

(VILLA-LOBOS, 1955)

1. A clave, que define quais notas cada linha ou espaço representam. Neste caso a clave é de Sol.

2. A armadura de clave, que indica as alterações a serem aplicadas às notas, indicando a tonalidade da música ou trecho.

3. A fórmula de compasso, que determina quantidade de tempos em cada compasso no número superior, e o valor de cada um desses tempos no inferior.

4. Uma sugestão de andamento, sugerindo qual a velocidade que a música deve ser tocada.

5. Esta e outras indicações são exemplos de notação específica aos instrumentos. Essa especificamente propõe que o trecho seja tocado com uma pestana na segunda casa do violão.

6. Uma notação de dinâmica de execução, orientando que a execução acelere um pouco a partir daquele trecho.

7. Outra notação de dinâmica, mas essa é relacionada a intensidade, propondo que o músico vá tocando mais forte a partir dela.

Há inúmeros outros símbolos além desses que transmitem informações sobre expressão musical, alterações de ritmo e andamento, dinâmica e técnicas específicas de execução. Por exemplo, o uso do pedal em partituras para piano ou sugestões de dedilhado em partituras para violão (MED, 1996).

Considerando a diversidade e complexidade dos elementos presentes nas partituras, uma alternativa para auxiliar na sua edição e composição é usar ferramentas digitais. Os sistemas atualmente disponíveis lidam com esses desafios por meio de interfaces visuais, nas quais os usuários interagem diretamente com os símbolos e suas posições na pauta, escolhendo quais elementos adicionar, remover ou modificar. Esse modelo de interface tem sido adotado, com pequenas variações, desde o surgimento dos primeiros softwares do gênero, como o *Finale* (1988)[1] e o *Sibelius* (1993)[2]. No entanto, esse tipo de edição apresenta alguns problemas intrínsecos ao seu funcionamento.

Observa-se na Figura 2, na interface do software MuseScore[3], alguns desses problemas, que são comuns à maioria dos editores utilizados atualmente. O grande conjunto de símbolos, figuras e suas variações que compõem a notação musical é exibido aos usuários por meio de menus e paletas, o que pode tornar a interface visualmente sobrecarregada e intimidante — especialmente para usuários inexperientes ou aqueles sem familiaridade com leitura musical. Além disso, o processo de entrada envolve a escolha de cada símbolo

[1] Finale. Disponível em: <https://www.finalemusic.com/>. Acesso em: 08 de abril de 2025.
[2] Sibelius. Disponível em: <https://www.avid.com/sibelius>. Acesso em: 08 de abril de 2025.
[3] MuseScore. Disponível em: <https://musescore.org/en>. Acesso em: 08 de abril de 2025.

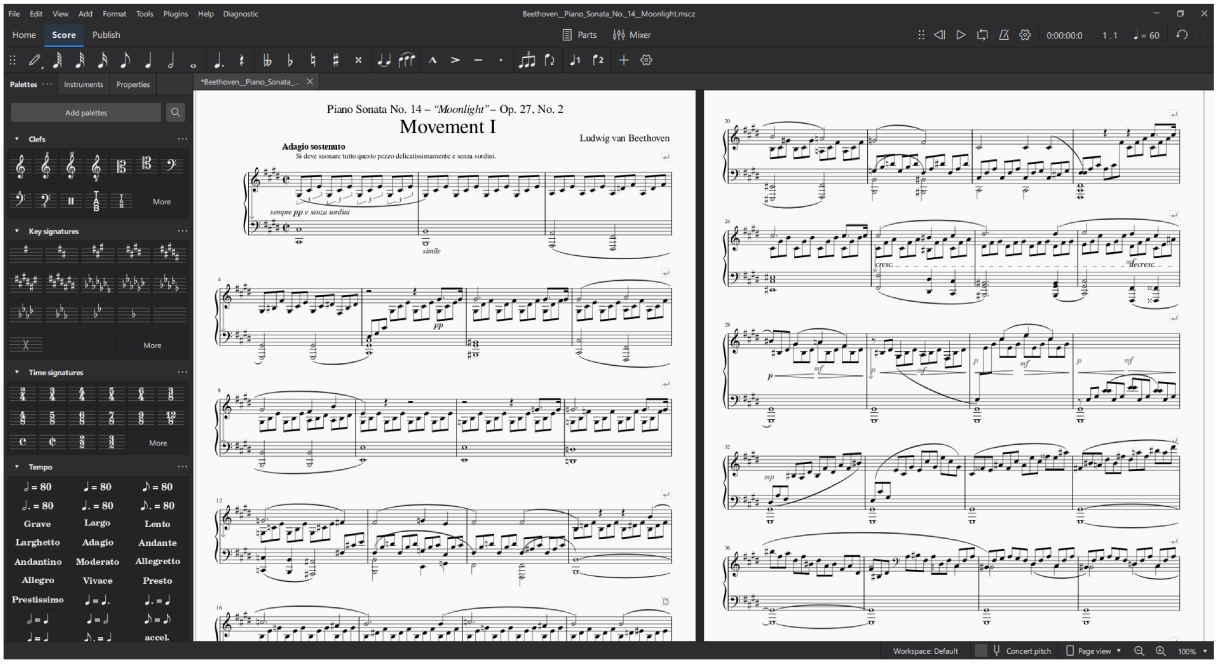

Figura 2 – Exemplo da interface do software MuseScore.

individualmente, o que, mesmo em pequenos trechos musicais, pode se tornar um processo demorado, repetitivo e tedioso.

Diante dos problemas observados, este trabalho propõe o desenvolvimento de uma ferramenta web com foco específico na facilidade e agilidade de uso. A escolha por uma plataforma baseada na web justifica-se pela ampla compatibilidade com dispositivos modernos, o que reforça o objetivo de praticidade. A disponibilização de um sistema com essas características pode contribuir de forma significativa para atividades como composição, arranjo, ensino musical e transcrição, atendendo tanto a profissionais quanto a alunos e músicos amadores em seu cotidiano.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolvimento de uma aplicação web interativa voltada para a edição de partituras musicais, com foco em praticidade e eficiência no processo de entrada e manipulação da notação musical. O sistema desenvolvido deve ser compatível com diferentes dispositivos, incluindo desktops e dispositivos móveis. Deve explorar os recursos disponíveis no ambiente de desenvolvimento web para viabilizar a exibição visual clara da partitura e seus elementos, assim como oferecer reprodução sonora e métodos otimizados de entrada musical, visando atender tanto usuários iniciantes quanto músicos experientes.

1.1.2 Objetivos Específicos

Para o desenvolvimento da aplicação web interativa proposta, foram definidos os seguintes objetivos específicos. Cada objetivo contempla etapas práticas do desenvolvimento, bem como a investigação e adoção de bibliotecas, frameworks e abordagens que viabilizem sua implementação de forma eficiente. Os objetivos específicos são:

- ❏ Projetar e implementar uma interface interativa para permitir a inserção e edição dos elementos musicais.
- ❏ Integrar bibliotecas especializadas para a exibição visual precisa da notação musical com base nos dados de entrada fornecidos pelo usuário.
- ❏ Incorporar funcionalidades de reprodução sonora da partitura, possibilitando ao usuário ouvir o resultado de sua entrada em tempo real.
- ❏ Pesquisar e testar diferentes métodos de entrada e edição musical, avaliando sua viabilidade e implementando as soluções mais eficazes no sistema.

1.2 Estrutura do Trabalho

Os capítulos seguintes deste trabalho está organizado da seguinte forma. O Capítulo 2 aborda os conceitos fundamentais relacionados ao desenvolvimento web e à teoria musical, além de revisar brevemente trabalhos correlatos. O Capítulo 3 descreve as tecnologias e a metodologia utilizadas para a criação do editor de partituras. Em seguida, o Capítulo 4 apresenta o sistema em termos de sua arquitetura, implementação e funcionalidades. Por fim, o Capítulo 5 traz as considerações finais e sugere direções para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

A fundamentação teórica deste trabalho abrange as áreas que sustentam tanto o aspecto técnico quanto o conceitual da proposta. Por um lado, são considerados elementos do desenvolvimento web e da engenharia de software, com ênfase em arquitetura de aplicações interativas, gerenciamento de estado e questões de usabilidade da interface. Por outro, exploram-se conceitos da teoria musical e dos sistemas de notação musical ocidental moderna, fundamentais para a implementação adequada e funcional dos elementos musicais no ambiente digital. A integração entre essas duas áreas é essencial para o desenvolvimento de uma ferramenta que seja tecnicamente robusta e, ao mesmo tempo, musicalmente precisa e intuitiva para o usuário.

2.1 Desenvolvimento Ágil

Os princípios do desenvolvimento ágil priorizam entregas incrementais, mantendo um protótipo funcional desde cedo, bem como a colaboração com usuários e a adaptação contínua ao longo do processo. Segundo Sommerville (2011), metodologias ágeis favorecem ciclos curtos de desenvolvimento e respostas rápidas a mudanças de requisitos, promovendo maior flexibilidade e eficiência na criação de sistemas interativos. Na página 92, o autor introduz um questionário que aborda aspectos a serem levados em conta ao adotar o desenvolvimento ágil, contribuindo para a avaliação de se este método de desenvolvimento seria realmente adequado.

1. A comunicação entre os membros da equipe é um dos elementos fundamentais neste tipo de desenvolvimento, então o autor propõe que a escala do sistema seja considerada, pois projetos em larga escala necessitariam de uma equipe maior. O sistema proposto neste trabalho é de pequena escala, com um único desenvolvedor, portanto a adoção do desenvolvimento ágil não seria um problema.

2. Também é indicado que o tipo de sistema a ser desenvolvido seja analisado, pois aplicações com requerimentos complexos de comunicação e interação em tempo-real

entre sistemas impõem a necessidade de um planejamento mais cuidadoso, dificultando ciclos curtos de desenvolvimento. Este trabalho não possui outros componentes ou requisitos de comunicação como back-ends ou microsserviços.

3. A vida-útil do sistema é colocada pelo autor como um aspecto importante a ser considerado, pois gera a necessidade de uma documentação mais detalhada sobre o sistema, visando sua manutenibilidade por equipes subsequentes. Inicialmente foi considerado o desenvolvimento contínuo da aplicação apenas no contexto deste trabalho, sem a participação de outros desenvolvedores, o que não causaria os problemas levantados pelo autor.

4. O último ponto a ser considerado seria o desenvolvimento de um sistema sujeito a regulações e entidades externas, pois seria necessário, além de uma fase de planejamento, uma fase para a aprovação do sistema pelas entidades relevantes, tornando o desenvolvimento ágil inviável. Este trabalho não está sujeito a tais externalidades, portanto este ponto não se aplicaria.

A partir desta análise, a adoção do desenvolvimento ágil mostrou-se adequada ao projeto, especialmente devido à necessidade de constantes alterações na interface musical e ajustes baseados na usabilidade e experiência do usuário.

2.2 Usabilidade

De acordo com Henry, Abou-Zahra e White (2016), o World Wide Web Consortium (W3C) considera a usabilidade como um dos componentes da acessibilidade relacionando-a à capacidade de uma solução ser eficiente, eficaz e satisfatória para os usuários. Essa definição alinha-se à norma ISO 9241-11[1], a qual define a usabilidade como o grau em que um produto pode ser utilizado por um grupo específico de usuários procurando alcançar objetivos determinados de forma eficaz, eficiente e com satisfação, em um contexto de uso predefinido.

2.3 Desenvolvimento Web

A escolha da Web como plataforma de desenvolvimento deveu-se principalmente à sua compatibilidade quase universal com diferentes dispositivos e sistemas operacionais, contribuindo diretamente para os objetivos de acessibilidade e portabilidade da ferramenta. Nesta subseção, apresentam-se os principais conceitos e tecnologias envolvidos no desenvolvimento da aplicação, incluindo *frameworks*, linguagens e bibliotecas utilizadas na construção da interface e no gerenciamento da lógica da aplicação.

[1] ISO 9241-11:2018. Disponível em: <https://www.iso.org/standard/63500.html>. Acesso em: 04 de abril de 2025.

2.3.1 SPA — Single Page Application

Uma *Single Page Application* (SPA) é um tipo de aplicação web que opera inteiramente dentro de uma única página, sem a necessidade de recarregamento do navegador ou navegação para outro endereço a cada interação ou mudança de conteúdo [2]. Esse modelo baseia-se fortemente no uso de *scripts* para manipulação dinâmica do conteúdo da interface, por meio da adição, remoção ou atualização de elementos diretamente no DOM[3]. O desenvolvimento de SPAs é amplamente suportado por diversas bibliotecas e *frameworks* modernos — entre os quais destaca-se o React.

2.3.2 React

O *React*[4] é uma biblioteca JavaScript mantida pelo Facebook, amplamente utilizada para o desenvolvimento de aplicações web. Sua principal característica é o uso de uma estrutura baseada em componentes reutilizáveis, que permite a criação de interfaces dinâmicas e reativas com maior organização e manutenibilidade. Um de seus casos de uso mais comuns é o desenvolvimento de *Single Page Applications* altamente interativas, tornando-a especialmente adequado para este projeto. Além disso, o *React* oferece um ecossistema maduro e ferramentas integradas para o gerenciamento de estado, principalmente a ferramenta chamada *Redux Toolkit*[5], utilizada para o gerenciamento de estado e que facilitou bastante a gestão da informação inserida pelo usuário.

2.3.3 VexFlow

O *VexFlow*[6] é uma biblioteca JavaScript especializada na renderização de notação musical e tablaturas. Ela permite gerar elementos visuais de partituras diretamente em ambientes web, utilizando HTML5 e SVG para exibição gráfica. A biblioteca opera a partir de uma entrada estruturada — que define notas, pausas, claves, compassos, articulações, entre outros — e converte essas informações em uma partitura visual completa. Sua flexibilidade, combinada à capacidade de renderizar partituras em tempo real, torna o *VexFlow* uma solução ideal para aplicações interativas voltadas à edição e visualização de música, como a desenvolvida neste trabalho.

---

[2] SPA (Single Page Application). Disponível em: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>. Acesso em: 04 de abril de 2025.

[3] Introduction to the DOM. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction>. Acesso em: 04 de abril de 2025.

[4] React — The library for web and native user interfaces. Disponível em: <https://react.dev/>. Acesso em: 04 de abril de 2025.

[5] Redux Toolkit. Disponível em: <https://redux-toolkit.js.org/>. Acesso em: 04 de abril de 2025.

[6] VexFlow — JavaScript Music Notation and Guitar Tablature. Disponível em: <https://www.vexflow.com/>. Acesso em: 04 de abril de 2025.

2.3.4 Tone.js

Para lidar com aspectos relacionados à reprodução e captura de áudio, foi encontrada uma biblioteca chamada Tone.js[7], que oferece uma camada de abstração sobre a API de Áudio da Web (Web Audio API), simplificando o desenvolvimento de aplicações que visam utilizar tais funcionalidades. O Tone.js também oferece suporte à entrada de áudio, viabilizando o reconhecimento de informações musicais por meio de som capturado, o que amplia as possibilidades de interação e usabilidade do sistema.

2.4 Teoria e Notação Musical

Para o desenvolvimento das ferramentas propostas neste trabalho, é fundamental dominar os conceitos básicos da teoria musical, especialmente no que diz respeito à teoria e notação utilizada do sistema ocidental moderno. Esses conhecimentos são essenciais para garantir que os símbolos, estruturas e relações musicais sejam representados de forma precisa e funcional na interface da aplicação.

As ferramentas técnicas citadas no subcapítulo anterior não possuem funcionalidades abrangentes o suficiente para garantir que a partitura seja exibida de forma musicalmente correta, como, por exemplo, garantir que a quantidade de notas em um compasso não exceda a duração estabelecida pela fórmula de compasso. Estes problemas são resolvidos programando funções que processam a entrada de dados e garatem que estejam sempre em um estado musicalmente correto.

Em relação à teoria da música, Med (1996) apresenta de forma abrangente os principais conceitos e elementos fundamentais da teoria e da notação musical ocidental. Da mesma forma, Chediak (2009) complementa essa base teórica com explicações mais práticas do assunto. Ambas as referências foram fundamentais para as etapas do projeto que envolvem a implementação de lógicas baseadas nas regras da escrita musical, garantindo a fidelidade teórica da aplicação desenvolvida.

2.5 Trabalhos Relacionados

Nas últimas décadas as inovações tecnológicas, em geral, vem sendo cada vez mais empregadas em diversos contextos musicais, como na educação, na construção e desenvolvimento de instrumentos, na gravação e trabalhos em estúdio, na composição e arranjo de obras e diversos outros. Nesta subseção, são brevemente listados alguns trabalhos com objetivos similares ou aspectos relacionados ao projeto proposto.

Considerando uma aplicação para Web com elementos musicais, Bergamo (2015) desenvolveu uma aplicação com um conjunto de instrumentos virtuais, jogos, e aplicativos

[7] Tone.js. Disponível em: https://tonejs.github.io/. Acesso em: 04 de abril de 2025.

para serem utilizados no contexto da musicoterapia, implementados para utilizar diversos periféricos de entrada como *Webcam*, *Gamepads*, *Touchpads*, microfone, entre outros. Para isso, ele utilizou as linguagens comuns para desenvolvimento Web, que incluem HTML, CSS, e *Javascript*, aproveitando de bibliotecas como jQuery, howler.js, e outros recursos de áudio e vídeo disponibilizados pelos próprios navegadores.

Já Carvalho (2020) elaborou uma ferramenta que adapta um software chamado MMA *(Music MIDI Acompaniment)* para ser utilizado através de uma interface Web mais amigável. O MMA é um programa de código aberto que funciona via linha de comando, escrito na linguagem Python. Considerando as dificuldades relacionadas à utilização de programas em linha de comando, o autor considerou a suposição de que uma ferramenta visual seria muito mais acomodadora para os usuários comuns. A aplicação foi desenvolvida com uma estrutura *full-stack*, em que o software gerador de acompanhamentos ficava no *back-end*, desenvolvido utilizando o *framework Flask*, e enviava os resultados para o *front-end*, desenvolvido em React.

Focado em uma aplicação voltada para estudantes iniciantes, Morais (2022) propôs uma aplicação Web desenvolvida para educação musical. Como entrada, os estudantes fornecem um conjunto simples de acordes e, como saída, a ferramenta gera uma análise harmônica tonal descrevendo os acordes, graus e prováveis cadências. A ferramenta foi desenvolvida com a linguagem PHP no *back-end* e Javascript no *front-end* com outras bibliotecas como *Bootstrap* e *Tone.js*; as funcionalidades de análise foram implementadas através da leitura de um arquivo XML que contém as definições mais comuns de intervalos, escalas e acordes.

Em relação à geração de partituras, Ramos (2015) propôs um sistema para web para transcrição automática de partituras em tablaturas. Para isso, utilizou o algoritmo *Ant Colony Optimization* (BLUM, 2005) para lidar com o problema do surgimento de dedilhados inviáveis para execução quando se convertem diretamente as notas de uma partitura para tablatura.

Por fim, Luz e Camargo (2019) propuseram um software para a criação de partituras via musicografia em braille, com o objetivo de promover a inclusividade de músicos cegos ou com baixa visão. A proposta consiste principalmente na exploração e discussão de tecnologias assistivas para uma musicalização inclusiva e para a criação de partituras. Além disso os autores discutem um pouco sobre tecnologias de desenvolvimento Web, como PHP, Javascript e bancos de dados em MySQL.

Na tabela 1, se observa uma comparação entre as tecnologias utilizadas por cada um dos trabalhos. Uma das principais similaridades entre este trabalho e os trabalhos relacionados é a escolha da biblioteca Tone.js para se trabalhar com áudio no navegador. Entre estes trabalhos, dois utilizaram frameworks para o *front-end*, que foram o React e AngularJS; os outros trabalharam com páginas em HTML. Uma das principais diferenças entre os trabalhos e este foi a implementação de um *back-end*, através dos quais os autores implementaram as funcionalidades necessárias para cumprir seus objetivos. Na aplicação que foi desenvolvida neste trabalho, as funcionalidades necessárias foram implementadas utilizando somente o Typescript e o React, sem que fosse necessário estabelecer uma arquitetura de cliente-servidor.

Trabalhos	Tecnologias			
	Linguagem	Front-end	Back-end	Áudio
Este trabalho	Typescript	React	-	Tone.js
Bergamo (2015)	Javascript	HTML, CSS, jQuery	-	howler.js
Carvalho (2020)	Python	React	Flask	-
Morais (2022)	PHP	HTML, Bootstrap	PHP	Tone.js
Ramos (2015)	Java	AngularJS	Jersey	Tone.js
Luz e Camargo (2019)	PHP	HTML	PHP	

Tabela 1 – Comparação entre as tecnologias utilizadas pelos trabalhos relacionados.

Capítulo 3

Metodologia

Este capítulo descreve os materiais, ferramentas e métodos utilizados para o desenvolvimento da ferramenta, além de detalhar o processo de desenvolvimento adotado.

3.1 Ambiente de Desenvolvimento

O primeiro passo para o desenvolvimento da ferramenta foi a configuração de um ambiente de programação voltado para aplicações web. Considerando que o *framework* React foi adotado como base do projeto, tornou-se necessário preparar um ambiente compatível, que fornecesse as ferramentas adequadas para seu uso eficiente. Desde o início, o projeto e o código foram desenvolvidos na língua inglesa devido às convenções da área e às estruturas semânticas da língua, que tornam o código mais fácil de ser compreendido.

3.1.1 Node.js e Vite

O Node.js[1] oferece uma plataforma que permite a execução de código JavaScript fora de navegadores. A configuração do Node.js foi uma das primeiras tarefas, sendo necessária porque *frameworks* modernos como o React dependem de um processo de compilação, no qual o código-fonte — frequentemente escrito com sintaxes mais avançadas e recursos adicionais (neste caso específico, a sintaxe TSX) — é convertido em JavaScript puro, apto para ser executado no navegador. Esse processo é gerenciado pelo Vite[2], uma ferramenta de compilação que, além de suportar projetos React, oferece funcionalidades importantes como um servidor de desenvolvimento local, recarregamento automático das alterações feitas no código, suporte nativo ao uso de TypeScript, além da organização das configurações de todas as ferramentas.

Todas as dependências do projeto, incluindo *frameworks*, ferramentas, bibliotecas e outras, são gerenciados pelo NPM — Node Package Manager[3], possibilitando que a apli-

[1] Node.js. Disponível em: <https://nodejs.org/>. Acesso em: 04 de abril de 2025.
[2] Vite. Disponível em: <https://vite.dev/guide/>. Acesso em: 04 de abril de 2025.
[3] NPM. Disponível em: <https://www.npmjs.com/>. Acesso em: 04 de abril de 2025.

cação seja instalada e executada em qualquer ambiente. O controle destas dependências e dos comandos necessários para compilar e executar o aplicativo é feito através do arquivo `package.json`, localizado no diretório raiz do projeto.

3.1.2 Versionamento e CI/CD

O controle de versionamento do projeto foi realizado por meio do GitHub, no repositório disponível em: <https://github.com/carloshrm/part-tcc>. Para garantir a padronização das mensagens de *commit* e facilitar o entendimento das alterações ao longo do desenvolvimento, adotou-se o uso do padrão Conventional Commits[4].

A partir do primeiro protótipo funcional, foi configurado um processo de Integração Contínua e Entrega Contínua (CI/CD)[5] utilizando a plataforma Azure em conjunto com o *GitHub Actions*. Isso automatizou a compilação do projeto a cada nova alteração enviada ao repositório, realizando a publicação da aplicação no contêiner previamente configurado na Azure. Dessa forma, a distribuição da aplicação passou a ocorrer de maneira contínua e automatizada, mantendo o sistema sempre atualizado. O projeto foi publicado no endereço: <https://carloshrm-partituras.azurewebsites.net/>.

A adoção de um processo de CI/CD alinha-se aos com os princípios do desenvolvimento ágil, pois permite entregas frequentes, incrementais e com menor risco de falhas. A automação das etapas de compilação e implantação reduz o tempo entre o desenvolvimento e a disponibilização de novas funcionalidades, promovendo ciclos de feedback mais rápidos. Isso contribui diretamente para a melhoria contínua do produto e para uma resposta mais ágil às necessidades dos usuários finais, conforme a abordagem incremental e iterativa do desenvolvimento de software (SOMMERVILLE, 2011).

3.2 Desenvolvimento Web

Esta seção descreve as principais tecnologias e ferramentas utilizadas no desenvolvimento da aplicação, como o React, TypeScript, Vite, Redux Toolkit e bibliotecas específicas para notação e reprodução musical.

3.2.1 Typescript

O *TypeScript*[6] é uma linguagem de programação desenvolvida pela Microsoft que estende o JavaScript, incorporando recursos como tipagem estática e verificação de tipos

[4] Conventional Commits. Disponível em: <https://www.conventionalcommits.org/en/about/>. Acesso em: 04 de abril de 2025.

[5] CI/CD significa Continuous Integration e Continuous Delivery/Deployment. Esse modelo indica que alterações no código são automaticamente integradas, testadas e implantadas, permitindo atualizações frequentes no sistema já em ambiente de produção.

[6] TypeScript — JavaScript with syntax for types. Disponível em: <https://www.typescriptlang.org/>. Acesso em: 10 de abril de 2025.

em tempo de compilação. Isso possibilita a detecção precoce de erros durante o desenvolvimento, além de proporcionar melhor suporte por parte das ferramentas de edição de código. No contexto deste projeto, a adoção do *TypeScript* possibilitou a modelagem explícita das entidades envolvidas, contribuindo para a organização do código e reduzindo significativamente a incidência de erros relacionados ao gerenciamento do estado, especialmente em um fluxo de entrada de dados musicais potencialmente complexo.

3.2.2 React

Como proposto, a ferramenta foi desenvolvida como uma aplicação do tipo *Single-Page Application* (SPA), utilizando o framework *React*. Dentre os inúmeros recursos oferecidos por essa biblioteca, podemos destacar a capacidade de se construir componentes individuais, e se necessário, genéricos e reutilizáveis. Isso permitiu organizar a interface em unidades independentes, facilitando muito tanto a manutenção quanto a escalabilidade do código.

Além disso, o React fornece mecanismos abrangentes para o gerenciamento do estado da informação na aplicação — recursos essenciais para este projeto, considerando que o estado representa toda a entrada de dados musicais fornecida pelo usuário. Através dos *hooks* de estado e de efeito (*useState* e *useEffect*), foi possível programar a aplicação para reagir dinamicamente às alterações na informação inserida, atualizando em tempo real a exibição gráfica da partitura. Essa abordagem garantiu uma experiência de usuário fluida e responsiva, simplificando também o controle da lógica interna da aplicação.

Além disso, por meio destes *hooks*, que a lógica musical foi implementada. Foram feitas funções que rodam quando acontece a entrada de dados, então comparam a entrada com o estado atual, e tem como saída um estado que seja musicalmente coeso. Esse mecanismo foi utilizado para o controle de divisão de compassos, duração de notas, acidentes, alterações e adições de notas, alterações de compasso e as outras diversas operações disponíveis ao usuário que alteram os elementos musicais.

3.2.3 Gerenciamento de Estado

O estado dos dados em um software representa as informações armazenadas em determinado momento durante a execução do programa. Ele reflete a condição atual dos dados que o sistema tem disponível para exibir informações ao usuário ou executar operações. Esse estado pode mudar conforme o usuário interage com o sistema ou à medida que eventos internos ocorrem.

Para o gerenciamento do estado, utilizou-se uma biblioteca chamada Redux-Toolkit[7]. Esta biblioteca fornece ferramentas para organizar, manter, e manipular os dados na aplicação. Em seguida, criou-se uma *store* central, que representa o conjunto completo de

[7] Redux-Toolkit. Disponível em: <https://redux-toolkit.js.org/>. Acesso em: 04 de abril de 2025

informações do sistema, contendo diferentes *slices*, que representam conjuntos distintos destas informações. Foram implementados dois *slices*: um para a informação musical, outro para as configurações de áudio. Dentro de cada *slice* são definidos *reducers*, que são as funções que alteram os dados, e *selectors*, que extraem partes relevantes destes dados para serem utilizados nos componentes. Um exemplo de um *reducer* importante que foi criado é o `setNote`, que tem uma nota musical como entrada, e adiciona esta nota á um vetor de notas, mas que ao fazer isto, processa todas as notas para garantir que não tenha notas sobrando ou faltando em um compasso.

Desenvolveu-se um *selector* que organiza um vetor de notas em compassos, respeitando a fórmula de compasso definida. Dessa forma, qualquer componente que precisar desta informação pode acessá-la de forma fácil. Esse modelo garante que os dados da aplicação venham sempre de um único lugar, promovendo consistência, previsibilidade e clareza no desenvolvimento, além de facilitar a reutilização e manutenção do código.

Uma outra biblioteca chamada *redux-persist*[8] foi utilizada como *middleware* na *store* para garantir a persistência dos dados inseridos pelo usuário. Essa biblioteca permite que o estado gerenciado pelo Redux seja automaticamente salvo em um mecanismo de armazenamento local do navegador, como o *localStorage*. Assim, mesmo que o usuário atualize a página, feche o navegador ou navegue para outra página, os dados anteriormente inseridos são restaurados automaticamente ao retornar. Essa funcionalidade melhora a experiência do usuário, evitando a perda acidental dos dados e contribuindo para uma interação mais confiável e fluida com a ferramenta.

Para ilustrar, a Figura 3 mostra o diagrama com interfaces utilizadas para modelar a maneira com que a informação é mantida no aplicativo. Nota-se que na figura os itens estão denominados como interfaces, refletindo como a implementação foi feita de fato, entretanto, de acordo com conceitos fundamentais de programação orientada a objetos, este tipo de objeto seria uma classe. Um dos principais motivos para se utilizar interfaces desta forma foi devido a uma forte recomendação da documentação de ambos React e Redux para evitar a utilização de objetos não-serializáveis em *stores* e *hooks* [9].

Considerando a denominação tecnicamente correta, a classe principal, *MusicDataState*, é a que descreve a estrutura dos dados musicais na *store*. Nesta classe temos o parametro *selectedNote* para armazenar a nota que o usuário selecionou para edição na partitura, *hoverNote* para qual nota está realçada, a fórmula de compasso em *keySignature*, o título da partitura no *title*, e finalmente, em *clef*, qual a clave selecionada. A fórmula de compasso é representada pela classe *TimeSignature*, na qual o número de tempo fica como *beats*, e o valor de cada tempo como *value*. A interface *Note* descreve como as notas foram armazenadas, com a duração em *duration* e *voice* para indicar a

[8] redux-persist. Disponível em: <https://github.com/rt2zz/redux-persist>. Acesso em 04 de abril de 2025

[9] Guia de Boas Práticas do Redux. Disponível em: <https://redux.js.org/style-guide/#do-not-put-non-serializable-values-in-state-or-actions>. Acesso em: 04 de abril de 2025.

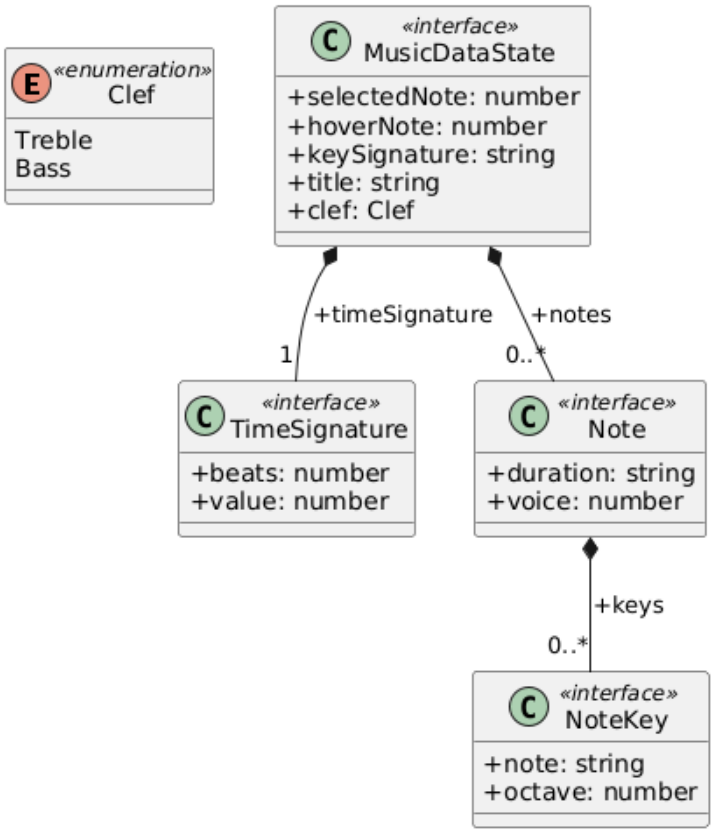

Figura 3 – Diagrama de classes do projeto.

qual voz pertence. O sistema para inserção de múltiplas vozes diferentes foi planejado, mas não foi implementado. A parte importante da *Note* é que ela tem uma *NoteKey*, que declara qual é aquela nota especificamente no parametro *note*, e em qual oitava no *octave*. O enúmeravel *Clef* foi utilizado para controlar quais os nomes válidos para a configuração de claves no sistema.

3.3 Notação Musical

A partir das interfaces descritas anteriormente, foram definidos padrões internos para gravar e validar a informação musical contando as notas, fórmula de compasso, clave, etc. Ao definir esses padrões foi considerando a forma com que o VexFlow e o Tone.js recebem os dados, procurando minimizar quaisquer manipulações necessárias para garantir que tal informação seja compatível com os formatos de entrada de ambas as bibliotecas. Um dos primeiros problemas enfrentados na programação da lógica musical é como calcular as durações das notas. A Figura 4 mostra como funciona a subdivisão das figuras de duração na notação musical.

A figura musical mais longa utilizada na notação moderna é a primeira no topo, chamada semibreve, a qual tem sua duração definida como 1 unidade de tempo. A partir dela, cada figura subsequente tem metade da duração da figura anterior. Esta subdivisão

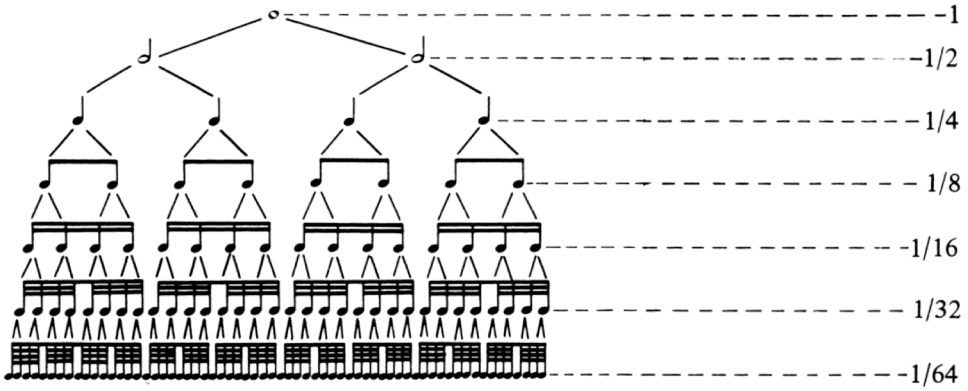

Figura 4 – Subdivisões das figuras musicais. Adaptada de Chediak (2009)

.

também está presente nas fórmulas de compasso, as quais definem um número de tempos e qual o valor da subdivisão de cada tempo. Como exemplo, podemos ver na Figura 5 como fica um compasso completo na fórmula `4/4`, no qual temos 4 notas da subdivisão 1/4.

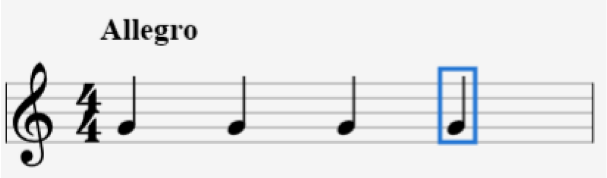

Figura 5 – Exemplo de um compasso completo na fórmula de 4/4

No projeto, as durações das notas foram implementadas utilizando apenas os números inteiros que representam as frações; por exemplo, uma semínima foi definida com a duração '4'. Vale ressaltar que frações e números decimais foram evitados ao máximo, pois, nos protótipos iniciais, erros causados por imprecisão em operações com números decimais foram identificados.

Um exemplo de imprecisão decimal seria na soma de durações antes da adaptação para números inteiros. Ao somar `0.1 + 0.2`, temos o resultado `0.30000000000000004`, e se compararmos este resultado com o valor desejado, `0.3`, para executar um desvio condicional, o resultado seria falso e o programa não teria o comportamento esperado.

Para definir as alturas das notas, foi adotado o sistema de letras, que é comum na notação da música popular em cifras e em línguas de origem não-latina. As notas são representadas por letras de A até G, correspondendo às notas na escala diatónica, conforme ilustrado na Figura 6.

Definindo a forma de se representar alturas e duração, podemos construir um objeto completo que representa uma nota ou conjunto de notas na partitura. Ao utilizar as funções relacionadas a construção da exibição da partitura, foi necessário apenas formatar as notas como `nota/oitava`, e adicionar o caractere'r' para indicar a duração de uma

A — Lá
B — Si
C — Dó
D — Ré
E — Mi
F — Fá
G — Sol

Figura 6 – Representação das notas em cifra.

pausa. Já para a reprodução musical foi necessário formatar as durações com um caractere a mais: `duração + n`. A Figura 7 exemplifica um trecho de notas em código e como fica o resultado visual gerado a partir dela.

```
notes: [
  {
    keys: [
      { note: "e", octave: 5, },
      { note: "g", octave: 4, },
      { note: "a", octave: 4, },
      { note: "c", octave: 4, },
    ],
    duration: "4",
    voice: 1,
  },
```

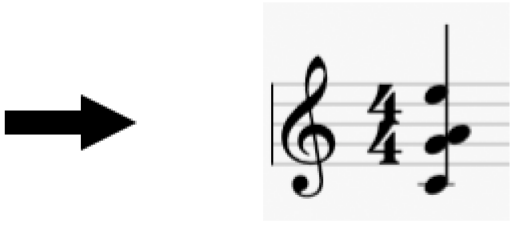

Figura 7 – Exemplo demonstrando a relação entre código e notas na pauta.

Um dos principais desafios no desenvolvimento da aplicação foi garantir a exibição precisa e visualmente fiel da notação musical dentro de um ambiente Web. Para resolver esse problema, foi adotada a biblioteca *VexFlow*[10], escolhida principalmente por sua flexibilidade, ampla adoção na comunidade e documentação detalhada.

No entanto, a biblioteca se limita à exibição gráfica dos símbolos musicais, não oferecendo suporte nativo às regras lógicas da teoria musical — como divisão de compassos, quebras de linha, ou agrupamento coerente de figuras rítmicas. Por conta disso, foi necessário implementar manualmente a lógica responsável por processar os dados musicais inseridos pelo usuário e garantir a coesão da notação.

Essa lógica foi implementada através dos *reducers* mencionados anteriormente, mantendo um controle no qual as partes específicas da informação musical podem ser alteradas somente por eles, garantindo a consistência na entrada da informação. A partir das adições e alterações feitas pelo usuário, uma função de processamento é executada dentro do

[10] VexFlow — JavaScript Music Notation and Guitar Tablature. Disponível em: <https://www.vexflow.com/>. Acesso em: 08 de abril de 2025.

reducer, garantindo que os compassos e fórmula de compasso sejam sempre válidos, e que os acidentes e a armadura de clave estejam consistentes.

Quando as alterações são efetivadas na *store*, o *hook* do React é ativado, executando uma função que converte os dados musicais em estruturas compatíveis com o VexFlow, que por sua vez realiza a renderização final da partitura utilizando a API *Canvas* do HTML. Esse processo garante que qualquer modificação feita pelo usuário seja refletida dinamicamente na partitura exibida.

3.3.1 Reprodução Sonora

Para a reprodução sonora, foi utilizada a biblioteca *Tone.js*[11], que fornece uma série de abstrações sobre a API de áudio nativa dos navegadores, tornando sua utilização significativamente mais acessível. Uma das principais funcionalidades exploradas foi o sintetizador polifônico disponibilizado pela biblioteca, que permite a execução simultânea em tempo real. Embora a aplicação não utilize sons "reais" de instrumentos musicais, uma grande vantagem dessa abordagem é a eliminação da necessidade de arquivos de áudio externos, como *samples* de instrumentos, que poderiam aumentar consideravelmente o tempo de carregamento inicial da aplicação devido ao tamanho dos arquivos.

3.4 Usabilidade

Depois que um protótipo com as funcionalidades básicas foi implementado, o projeto começou a ser estilizado levando em considerações as convenções que ditam as práticas para a experiência do usuário ao utilizar uma aplicação para web. Os componentes foram estilizados considerando uma separação clara e intuitiva entre cada um, empregando uma paleta de cores que oferece um contraste de pelomenos 4.5:1 entre o texto e o plano de fundo. Estas são orientações do WCAG 2.1 estabelecido pela W3C[12]. Foi utilizada uma ferramenta automática chamada Browserstack[13] para se avaliar a aderencia da ferramenta às convenções, obtendo um resultado de 91 em 100, o que indica um bom índice de usabilidade.

3.4.1 Interface e estilização

Uma biblioteca chamada *styled-components*[14] foi utilizada para a estilização da aplicação. Esta biblioteca permite que os estilos CSS sejam definidos diretamente nos arquivos

[11] Tone.js. Disponível em: <https://tonejs.github.io/>. Acesso em: 04 de abril de 2025.
[12] WCAG 2.1. Disponível em: <https://www.w3.org/TR/WCAG21/>. Acesso em 09 de maio de 2025.
[13] Browserstack. Disponível em: <https://www.browserstack.com/>. Acesso em 09 de maio de 2025.
[14] styled-components. Disponível em: <https://styled-components.com/>. Acesso em: 04 de abril de 2025.

JavaScript ou TypeScript, vinculando cada estilo a um componente específico. Isso proporciona um encapsulamento do código dos elementos visuais, evitando conflitos de estilos sem que fosse necessário nos preocuparmos com padrões de nomeação de classes CSS.

Além disso, o *styled-components* oferece suporte para propriedades dinâmicas, permitindo a alteração de regras de estilo em tempo real com base nos dados fornecidos pelo usuário ou pelo estado da aplicação. Os elementos estilizados de cada componente React foram organizados em arquivos separados chamados *styles.ts*, localizados dentro de suas respectivas pastas, o que contribui para a clareza, organização e manutenibilidade geral do projeto.

Além do *styled-components*, também foi utilizada a biblioteca Ant Design[15], que fornece uma coleção de componentes de interface pré-programados e pré-estilizados, como botões, campos de entrada, menus, modais, entre outros. Esses componentes já vêm implementados com base em convenções de acessibilidade, responsividade e usabilidade. A sua utilização agilizou significativamente o desenvolvimento da interface. A flexibilidade do *styled-components* foi aproveitada para personalizar visualmente os componentes do Ant Design conforme a identidade visual da aplicação, permitindo a criação de uma interface intuitiva e adaptável a diferentes tamanhos de tela.

Um detalhe importante foi a inclusão de uma fonte com uma licença de uso livre chamada 'Bravura' [16] no projeto. Esta fonte contém símbolos da notação musical e foi utilizada para inclusão de símbolos em botões e menus da interface.

3.4.2 Exportação em PDF e PNG

Também foi incluída uma biblioteca chamada jsPDF[17], que tem como função facilitar o processo de geração de arquivos PDF diretamente a partir do navegador. Essa funcionalidade permite que o usuário exporte a partitura criada em um formato amplamente utilizado para impressão e compartilhamento. Para gerar uma imagem em formato PNG da partitura, foi utilizada a API do elemento Canvas do HTML5, permitindo diretamente o *download* da partitura como imagem.

[15] Ant Design. Disponível em: <https://ant.design/>. Acesso em: 04 de abril de 2025.
[16] Fonte Bravura. Disponível em: <https://github.com/steinbergmedia/bravura/>. Acesso em: 04 de abril de 2025.
[17] jsPDF. Disponível em: <https://github.com/parallax/jsPDF>. Acesso em: 04 de abril de 2025.

Capítulo 4

Editor de Partituras

Neste capítulo, o aplicativo desenvolvido e suas funcionalidades serão demonstrados. Como o projeto foi desenvolvido como uma SPA, a aplicação possui uma única tela principal, na qual ocorrem todas as interações programadas.

Ao acessar a página pela primeira vez, o usuário verá a tela exibida na Figura 8. Podemos observar à esquerda uma série de menus. Ao clicar no botão que contém o nome de cada menu, eles podem ser minimizados ou maximizados; o estado inicial do aplicativo traz os menus de *Notas* e *Compassos* maximizados, por serem possivelmente os mais utilizados. Em destaque, são enumeradas na figura as seguintes partes:

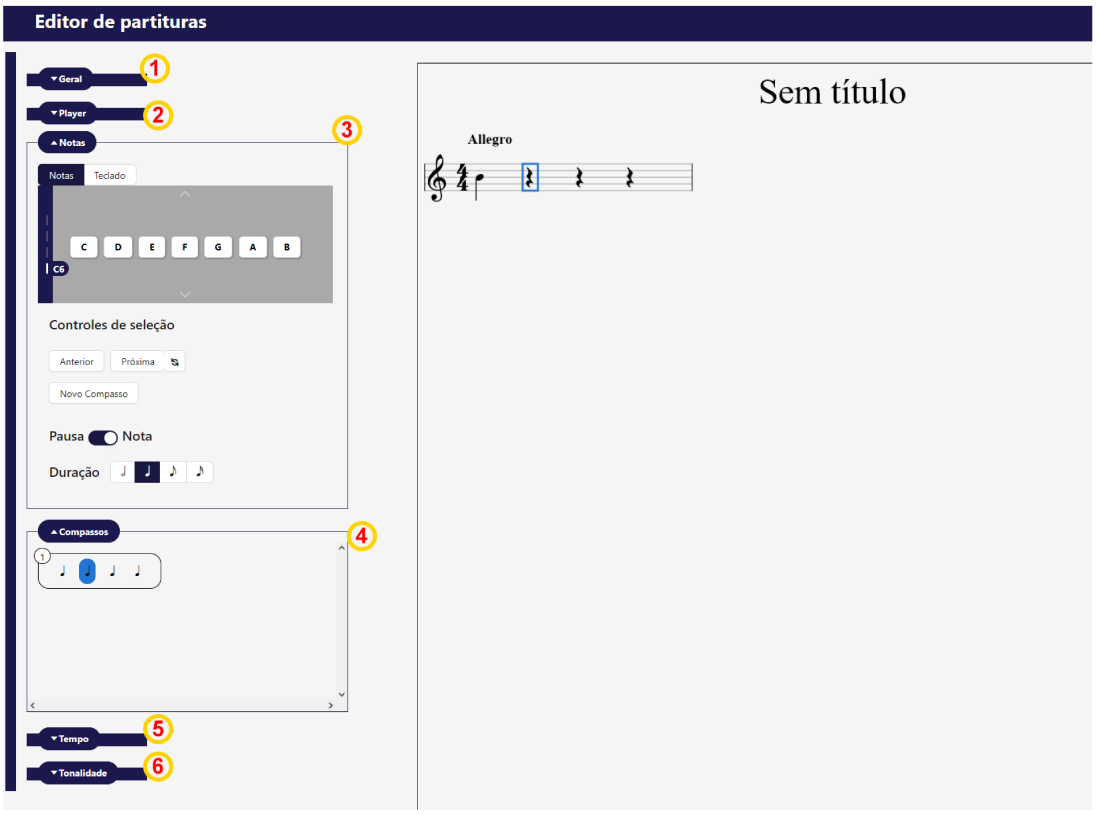

Figura 8 – Tela inicial do aplicativo.

1. O menu, exibido em detalhe na Figura 9, contém as configurações do título e da indicação de andamento na partitura, além de opções para salvar a partitura como PDF ou PNG, bem como um botão para limpar todas as alterações do usuário e retornar o aplicativo ao estado inicial.

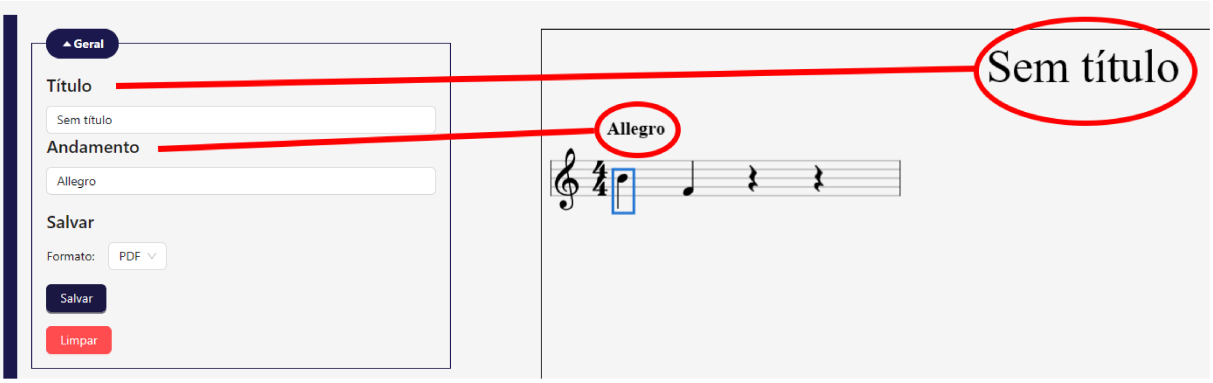

Figura 9 – Menu 'Geral'.

2. No menu *Player*, destacado na Figura 10, o usuário tem as opções relacionadas à reprodução sonora da partitura, podendo tocar toda a música escrita até o estado atual, parar a reprodução ou somente ouvir a parte selecionada no momento, incluindo também uma de volume e um botão para zerar o volume (modo 'mudo').

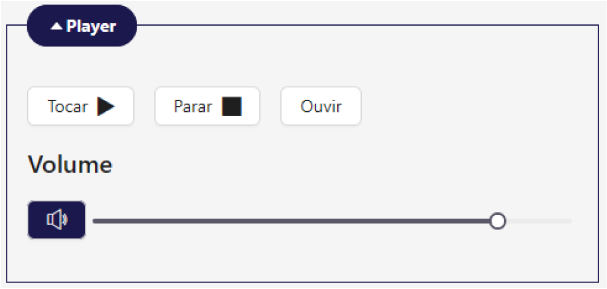

Figura 10 – Menu 'Player'.

3. O menu *Notas*, exibido em detalhes na Figura 11, é um dos principais da aplicação. O primeiro elemento dentro dele é a ferramenta para definição de notas na partitura, que pode ser alternada entre um submenu de *Notas* e um submenu *Teclado* virtual.

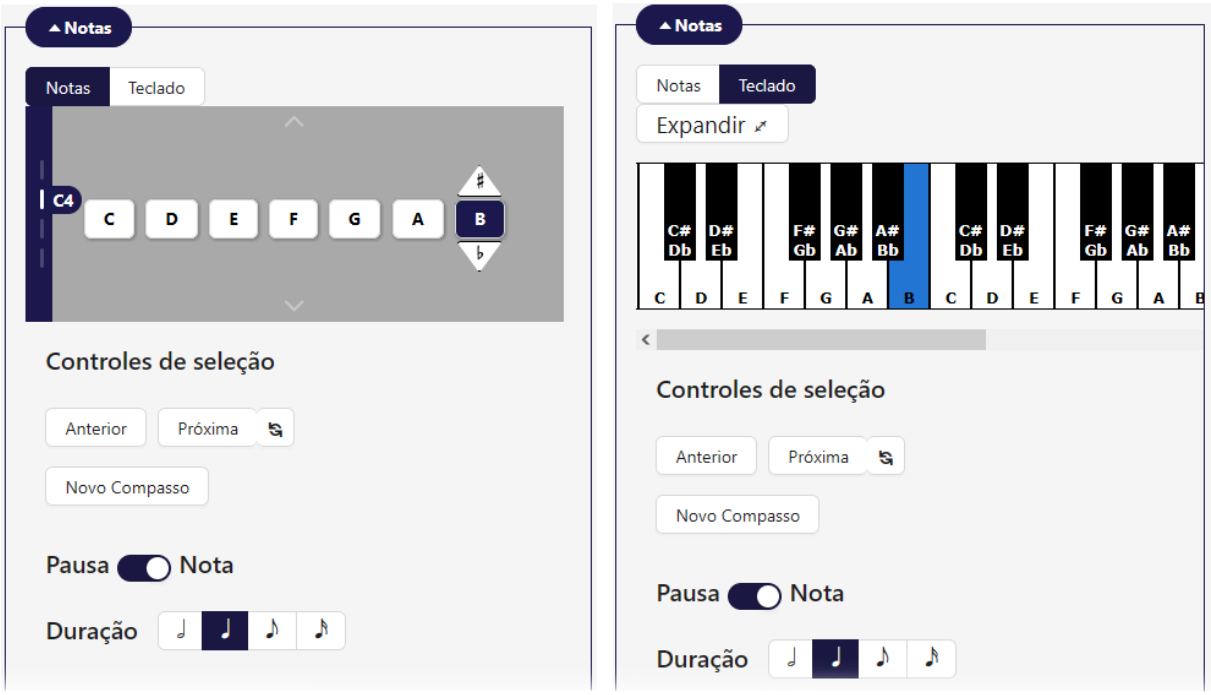

Figura 11 – Menu *Notas* com as opções de entrada *Notas* e *Teclado*

.

- ***Notas.*** À esquerda na Figura 11, um menu mais simples e intuitivo, permite a rolagem vertical para a seleção da oitava, onde o usuário pode clicar nas notas que deseja dentro daquela oitava; quando uma nota é clicada, o usuário pode escolher o acidente sustenido # ou bemol *b*.
- ***Teclado.*** À direita na Figura 11, há um teclado com 4 oitavas no qual o usuário pode marcar as notas que deseja, e caso prefira, um botão '*Expandir*' para maximizar o teclado para se estender sobre a pauta. A Figura 12 mostra o teclado virtual no modo expandido.

Além das ferramentas de entrada, temos o controle manual de seleção de notas e um botão para adicionar um novo compasso vazio (como mostrado nas partes inferiores da Figura 11); no controle para selecionar a próxima nota, temos um pequeno botão que ativa um modo que avança a seleção automaticamente, onde após uma edição, o aplicativo conta 2,5 segundos, e se não houver outra edição, ele avança a seleção para a próxima nota.

Temos também o controle para o usuário definir se ele quer uma nota ou uma pausa na posição selecionada. O o último controle neste menu é o de duração, que define a duração que a nota selecionada vai assumir na partitura.

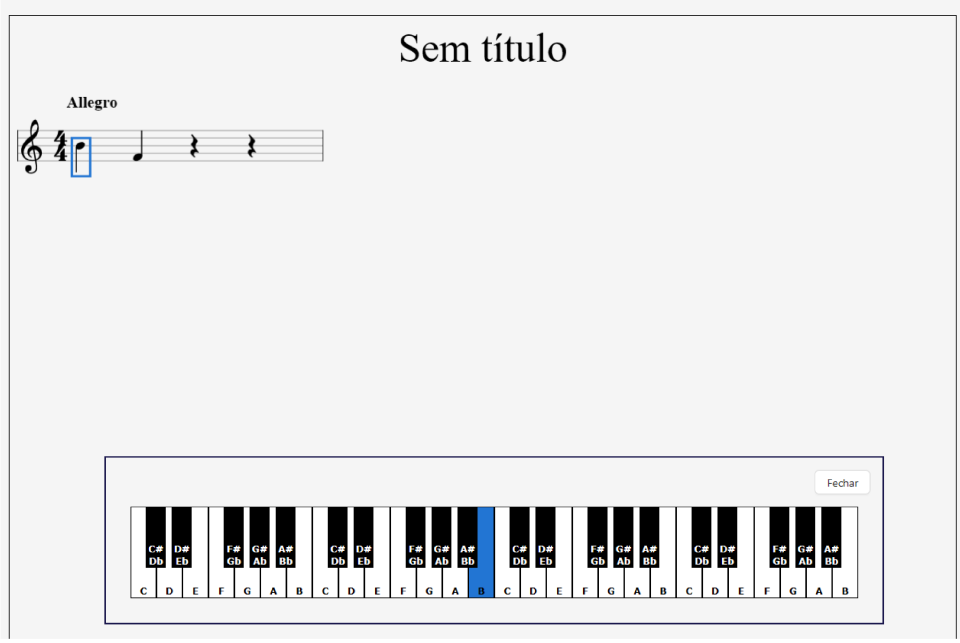

Figura 12 – Teclado no modo expandido.

Por fim, nota-se que atalhos de teclado foram implementados a partir das ações disponíveis neste menu. O usuário pode pressionar as teclas direcionais de direita e esquerda para avançar ou retroceder a seleção das notas, além de alterar a duração selecionada com os números de 1 a 4.

4. Em seguida, temos o menu *Compassos* que apresenta uma visualização resumida do conteúdo da partitura, auxiliando o usuário na navegação e na seleção das notas desejadas. Ao posicionar o mouse sobre as notas neste menu, elas são realçadas na partitura e vice-versa. Quando uma nota é selecionada, uma marcação é inserida tanto na partitura quanto no menu correspondente.

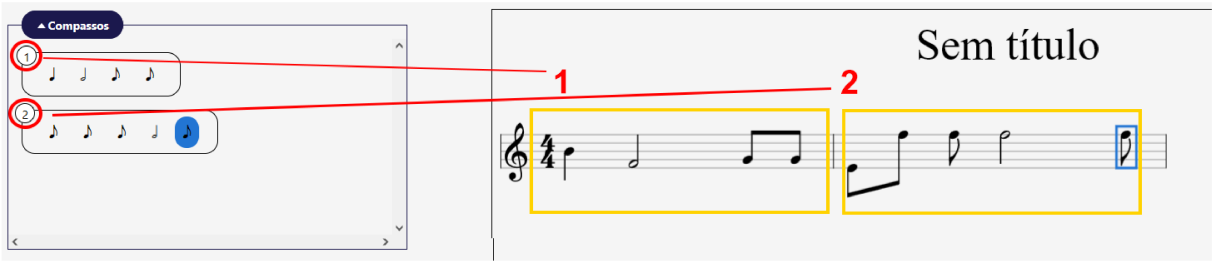

Figura 13 – Menu 'Compassos'

5. No menu *Tempo* são disponibilizadas opções relacionadas ao ritmo, como a fórmula de compasso e o andamento. A fórmula de compasso pode ser alterada a qualquer momento, e as informações entradas no aplicativo até o momento serão processadas para se adaptar à nova fórmula. O andamento configurado aqui também é utilizado como configuração na reprodução sonora.

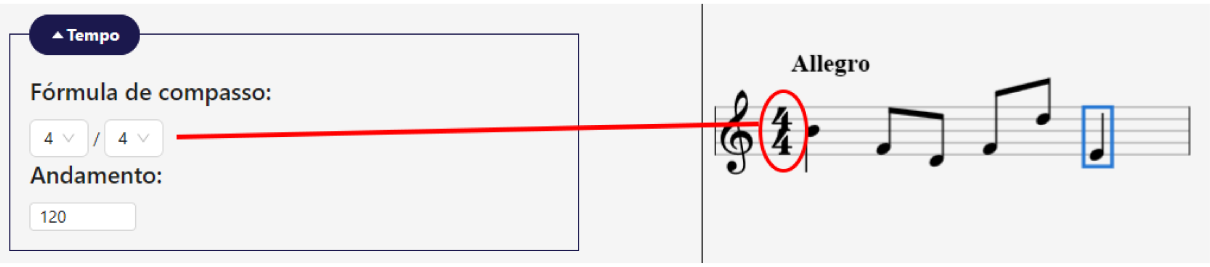

Figura 14 – Menu 'Tempo'

6. O último menu, *Tonalidade*, oferece os controles para o usuário escolher a armadura de clave, e também para alterar a clave entre Sol e Fá. O aplicativo foi programado para considerar a escolha da armadura de clave na exibição dos acidentes.

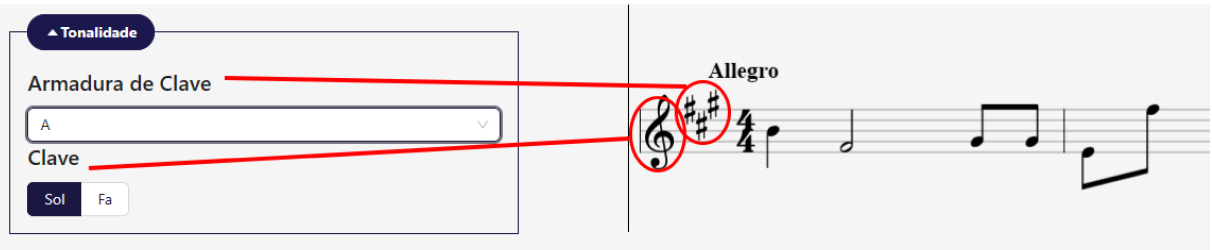

Figura 15 – Menu 'Tonalidade'

Capítulo 5

Conclusão

O desenvolvimento da aplicação proposta teve como objetivo central a criação de uma ferramenta acessível, prática e eficiente. Conforme os objetivos, buscou-se implementar uma solução que permitisse a inserção, visualização e reprodução sonora de elementos musicais de forma intuitiva, responsiva, e compatível com diferentes dispositivos. Ao longo do planejamento e da execução projeto surgiram desafios técnicos, muitos dos quais foram resolvidos através do uso das ferramentas descritas na fundamentação. Dentre estes desafios, os principais estavam relacionados à exibição visual da notação musical e à edição interativa dos dados musicais.

O React e bibliotecas encontradas para lidar com problemas específicos como o Redux Toolkit, e o Tone.js se mostraram corretas, oferecendo soluções abrangentes e eficientes. A biblioteca de notação musical, o VexFlow, se mostrou ser bastante flexível e completa em questão de funcionalidades, mas trouxe o problema de ter que manter a entrada de dados sempre musicalmente coesa através da implementação manual da lógica musical. Para desenvolvimentos futuros seria recomendável procurar alguma ferramenta que faça esta gestão, ou elaborar um projeto separadamente para resolver este problema específico. A organização do estado da aplicação, o uso de componentes reutilizáveis e a integração com ferramentas de CI/CD também contribuíram para a qualidade e a escalabilidade do sistema.

A aplicação desenvolvida cumpriu os objetivos específicos definidos: oferece uma interface clara e interativa para inserção e edição de notas musicais, exibe a partitura com fidelidade visual e conforme as regras da notação ocidental, e reproduz sonoramente a música em tempo real com resposta imediata às ações do usuário. Além disso, recursos como geração de PDF ou PNG, armazenamento de estado e uma interface responsiva e interativa oferecem uma melhor experiência ao usuário.

5.1 Principais Contribuições

Considerando a área de música em geral, o desenvolvimento de uma ferramenta que oferece edição de partituras na web demonstra a viabilidade da plataforma como base para diversos tipos de ferramentas musicais, demonstrando seus recursos e flexibilidade. Este trabalho pode possivelmente ser utilizado como uma base para ferramentas educacionais ou colaborativas. Para projetos educacionais, podem ser desenvolvidas seções pre-programadas para o ensino de leitura de partituras ou fundamentos da teoria musical, e pode ser implementadas funcionalidades para que dois usuários, possivelmente um aluno e professor, interajam com a mesma partitura.

Na área de desenvolvimento e programação, o trabalho também proporcionou aprendizados importantes quanto à aplicação de técnicas de desenvolvimento ágil, boas práticas de engenharia de software e a importância do planejamento da arquitetura de um software interativo. A organização em componentes através do React demonstra como uma ferramenta pode ser programada de forma escalável, flexível, e de fácil manutenção. Através deste tipo de projeto, os componentes podem ser programados isoladamente, serem individualmente testados, e então facilmente integrados na aplicação principal, facilitando esforços em equipe. A utilização do Typescript para aplicações web em um contexto onde diversas bibliotecas diferentes interagem entre si também foi importante pela possibilidade de se definir interfaces e modelar, desde o início do projeto, o fluxo de dados no aplicativo. Desta forma podemos simplificar e facilitar significantemente a utilização de diversas bibliotecas diferentes, garantindo a intercompatibilidade entre elas.

5.2 Trabalhos Futuros

Para trabalhos futuros, o projeto pode ser melhorado em diversas áreas. A primeira seria não somente a edição de partituras, mas também de tablaturas, que são bastante comuns para instrumentos como violão e guitarra. Podem ser implementados também métodos de importação e exportação utilizando formatos como MusicXML e MIDI, para os usuários conseguirem levar ou trazer seu trabalho de outras plataformas e programas. Podem ser implementadas funcionalidades de entrada de áudio, possibilitando que os usuários transcrevam música em tempo real através de solfejo.

Uma possibilidade interessante para a continuidade da ferramenta é a integração com técnicas de Inteligência Artificial. Através da informação das notas inseridas pelo usuário, seria possível aplicar modelos de aprendizado de máquina, como redes neurais recorrentes (RNNs), transformers ou modelos de linguagem adaptados ao contexto musical, para sugerir próximas notas, harmonias, e variações melódicas com base em padrões musicais aprendidos.

Como possibilidades de desenvolvimentos de outras ferramentas a partir deste pro-

jeto, podemos considerar a criação de plataformas de ensino e musicalização, e também ferramentas de colaboração em tempo real entre usuários. Outra possibilidade é a criação de uma espécie de biblioteca com conteúdo pre-programado como músicas e acompanhamentos.

Conclui-se, portanto, que os objetivos propostos foram alcançados com êxito, resultando em uma aplicação funcional, acessível e com grande potencial de uso prático e acadêmico, abrindo caminhos para futuras inovações na interseção entre tecnologia e educação musical.

Referências

BERGAMO, H. Desenvolvimento de aplicativos e jogos de música para utilização no campo da musicoterapia. **inCantare**, v. 6, n. 2, p. 73–96, 2015. Disponível em: <https://doi.org/10.33871/2317417X.2015.6.2.1268>. Citado na página 13.

BLUM, C. Ant colony optimization: Introduction and recent trends. **Physics of Life reviews**, Elsevier, v. 2, n. 4, p. 353–373, 2005. Disponível em: <https://doi.org/10.1016/j.plrev.2005.10.001>. Citado na página 14.

CARVALHO, H. N. de. **Ferramenta de acompanhamento musical: uma implementação com tecnologias web.** Dissertação (Mestrado) — Universidade de Brasília, Brasília, DF, 2020. Citado na página 14.

CHEDIAK, A. **Harmonia e Improvisação – Volume I**. São Paulo: Irmãos Vitale, 2009. ISBN 857407263X. Citado 3 vezes nas páginas 3, 13 e 21.

HENRY, S. L.; ABOU-ZAHRA, S.; WHITE, K. **Accessibility, Usability, and Inclusion**. 2016. Disponível em: <https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/>. Citado na página 11.

LUZ, F. D. V.; CAMARGO, V. G. Projeto de desenvolvimento de software para produção de partituras musicais utilizando a musicografia braille. **Textos para Discussão**, v. 1, n. 1, p. 007, 2019. Citado na página 14.

MED, B. **Teoria da Música**. Brasília, DF: Musimed, 1996. Citado 2 vezes nas páginas 7 e 13.

MORAIS, P. A. da R. Software analisador de harmonia na música popular: uma possível ferramenta para estudantes iniciantes. In: **Nas Nuvens... Congresso de Música 8**. [S.l.: s.n.], 2022. Citado na página 14.

RAMOS, A. S. **Um sistema web inteligente para a transcrição de partituras em tablaturas**. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, 2015. Citado na página 14.

SOMMERVILLE, I. **Engenharia de Software**. São Paulo, SP: Pearson Prentice Hall, 2011. Citado 2 vezes nas páginas 10 e 17.

SWAINE, M.; FREIBERGER, P. **Fire in the Valley: The Birth and Death of the Personal Computer**. 3rd. ed. Dallas, Texas: Pragmatic Bookshelf, 2014. 424 p. ISBN 978-1-937785-76-5. Citado na página 6.

VILLA-LOBOS, H. **Suite Popular Brasileira — 4. Schottish-Chôro**. Paris, França: Max Eschig, 1955. Citado na página 6.