Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Ferramenta de acompanhamento musical: uma implementação com tecnologias web

Autor: Hugo Neves de Carvalho
Orientador: Prof. Dr. Fábio Macêdo Mendes

Brasília, DF
2020

Hugo Neves de Carvalho

Ferramenta de acompanhamento musical: uma implementação com tecnologias web

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Fábio Macêdo Mendes

Coorientador: Prof. Joenio Marques da Costa

Brasília, DF

2020

Hugo Neves de Carvalho
Ferramenta de acompanhamento musical: uma implementação com tecnologias web/ Hugo Neves de Carvalho. – Brasília, DF, 2020-
54 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Fábio Macêdo Mendes

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2020.

1. acompanhamento musical. 2. web. I. Prof. Dr. Fábio Macêdo Mendes. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Ferramenta de acompanhamento musical: uma implementação com tecnologias web

CDU 0

Hugo Neves de Carvalho

Ferramenta de acompanhamento musical: uma implementação com tecnologias web

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, – Data da aprovação do trabalho:

Prof. Dr. Fábio Macêdo Mendes
Orientador

Prof. Dra. Carla Silva Rocha Aguiar
Convidado 1

Prof. Dr. Edson Alves da Costa
Convidado 2

Brasília, DF
2020

Dedico este trabalho a Música como forma de retribuição ao bem que me faz.

Agradecimentos

Agradeço a meus pais, irmãos, amigos e professores pelo apoio durante estes ano, que agora culminam na elaboração deste trabalho. Agradeço a Universidade de Brasília pelas oportunidades, experiências e conhecimentos adquiridos.

"I've seen things you people wouldn't believe.
Attack ships on fire off the shoulder of Orion.
I watched C-beams glitter in the dark near
the Tannhäuser Gate. All those moments
will be lost in time, like tears in rain.
Time to die."
(Roy Batty, Blade Runner - 1982)

Resumo

A tecnologia vem cada vez mais sendo utilizada para auxiliar a educação musical e os músicos em seus processos criativos. Um exemplo dessa parceria são os *softwares* que ajudam músicos em seus estudos práticos, durante execução de uma música ou exercício. O contexto deste trabalho discorre acerca do desenvolvimento de uma *interface* amigável para um programa que gera um acompanhamento musical baseado em entradas fornecidas pelo usuário como cifras musicais. Em termos arquiteturais a proposta apresentada utiliza o padrão *Web Full Stack*, com propósito de facilitar a distribuição da aplicação aos usuários. Está dividida em duas grandes partes, o *Frontend* responsável por coletar as informações necessárias do usuário e transferi-lás até a outra parte, o *Backend*. Esta baseada no *software MMA - Musical MIDI Accompaniment*, que gera um arquivo *.mid* contendo o acompanhamento desejado, que retorna e é executado e controlado diretamente no *browser*. A ferramenta neste trabalho desenvolvida é intitulada Bombaim, que é capaz de realizar estas atividades dentro do escopo proposto.

Palavras-chave: acompanhamento musical, web, instrumentos virtuais, MMA - Musical MIDI Accompaniment.

Abstract

Technology is increasingly being used to assist music education and musicians in their creative processes. An example of this partnership is software that helps musicians in their practical studies, while playing a song or exercising. The context of this work discusses the development of a friendly interface for a program that generates a musical accompaniment based on inputs provided by the user as musical figures. In architectural terms, the proposal presented uses the Web Full Stack standard, in order to facilitate the distribution of the application to users. It is divided into two major parts, the Frontend responsible for collecting the necessary information from the user and transferring it to the other part, the Backend. It is based on the MMA - Musical MIDI Accompaniment software, which generates a .mid file containing the desired accompaniment, which returns and is executed and controlled directly in the browser. The tool in this work developed is called Bombay, which is capable of carrying out these activities within the proposed scope.

Key-words: music accompaniment, web, virtual instruments, MMA - Musical MIDI Accompaniment.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
GUI	Graphical User Interface
MIDI	Musical Instrument Digital Interface
MMA	Music MIDI Accompaniment
NIST	National Institute of Standards and Technology - USA
JSON	- JavaScript Object Notation
HTML	- Hypertext Markup Language

Sumário

1 Introdução

Atualmente existem alguns *softwares* capazes de implementar o conceito de acompanhamento musical, substituindo os instrumentistas por instrumentos virtuais (THÉBERGE, 1997). Normalmente, estes softwares são utilizados como auxílio ao músico durante as seguintes atividades: composição, estudo, performace, entre outras. Atualmente, existe um considerável número de *softwares* contendo esta funcionalidade, mesmo que de forma limitada ou não explícitada.

Existem no mercado alguns exemplos de soluções proprietárias voltadas ao acompanhamento como *Band-in-a-Box* (BAND-IN-A-BOX, 2019) e *iReal* (IREAL, 2019). Nestas, o acompanhamento musical é gerado a partir de uma cifra fornecida pelo usuário e a seleção de um ritmo preexistente na ferramenta. No entanto, em ambos os *softwares* não existe a possibilidade que o usuário personalize as estruturas rítmicas da aplicação.

Alguns *softwares* livres como, por exemplo, *MMA - Music MIDI Accompaniment* (MMA, 2020), são capazes de realizar as atividades necessárias para criação do acompanhamento musical citadas anteriormente. No entanto, este não apresenta interface gráfica para interação com o usuário. Outros, mesmo não especializados no contexto de acompanhamento, também oferecem esta característica, por exemplo, o *Musescore* (MUSESCORE, 2019). Este possibilita a escrita tanto da harmonia quanto do ritmo, entretanto exige que seja representada utilizando notação convencional de partituras, o que tende a ser trabalhoso para usos casuais.

Os músicos, em geral, quando procuram programas para acompanhamento musical, geralmente encontram alguma das ferramentas anteriormente citadas. O principal objetivo desta categoria de *software* é fornecer uma experiência mais próxima possível do acompanhamento executado por músicos profissionais. Infelizmente, parte destas ferramentas foram construídas de forma que o usuário não pode customizá-los, principalmente no aspecto de criação de novos estilos musicais. Além disso, existem casos onde é necessário conhecimento em escrita de partituras, o que acrescenta dificuldade extra ao uso e que resulta em uma aplicação pouco atrativa para alguns usuários.

1.1 Objetivo Geral

Este trabalho tem como objetivo principal implementar uma aplicação *Web Full Stack*, de código aberto, que utilize o *software MMA* como gerador do acompanhamento e que apresente uma interface gráfica intuitiva para o usuário.

1.2 Objetivos específicos

- Desenvolver uma forma visual para inserção de harmonias, através de cifras e seleção do estilo de execução, para que seja possível reproduzir sonoramente um acompanhamento musical;
- Desenvolver uma forma visual para criação de novos estilos de reprodução das harmonias, bem como disponibilizar estas novidades na aplicação para uso coletivo.

1.3 Organização do trabalho

O presente trabalho está organizado em 4 capítulos. O Capítulo 2 traz o referencial teórico, conceitos e fundamentos a respeito de música e *softwares* de acompanhamento, desenvolvimento *web* e *Cloud Computing*. O Capítulo 3, apresenta metodologia utilizada para o planejamento e validação das ideias. O Capítulo 4 contém o desenvolvimento da proposta de trabalho, bem como os resultados gerados e algumas considerações sobre estes. No Capítulo 5 são finalizadas as discussões e apresentadas propostas de trabalhos futuros. Ao final, apresentam-se as referências bibliográficas utilizadas na elaboração deste trabalho.

2 Referencial Teórico

2.1 Conceitos musicais

Uma música pode ser entendida como associação sucessiva de sons e do silêncio, utilizando diversas metodologias e técnicas. A composição artística de uma música envolve as seguintes partes: melodia, harmonia e ritmo. A melodia é conjunto de sons dispostos de forma sucessiva, já a harmonia os dispõe de forma simultânea, o ritmo fornece a ordem e proporção que a melodia e harmonia se adequam. (MED, 1996).

O acompanhamento musical, Figura 1, é a junção da harmonia e ritmo, com objetivo de fornecer sustentação para a execução de uma melodia. A harmonia geralmente é composta por um sequenciamento de acordes, conhecido como progressão, já o aspecto rítmico é responsável pela geração da pulsação da música.

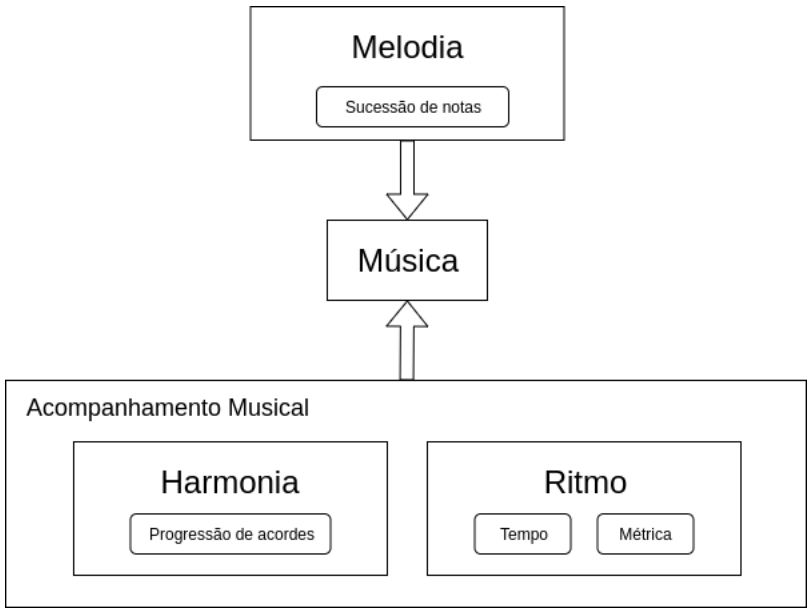


Figura 1 – Organização de uma música

Geralmente, cada instrumento apresenta uma das funções anteriomente citadas, sendo o acompanhamento comumente reproduzido em conjunto por instrumentos que não estão conduzindo a melodia. Por exemplo, uma banda formada por Bateria, Contrabaixo, Guitarra e Voz pode ser entendida da seguinte forma: a Voz conduz a linha melódica, a Bateria preocupa-se com a execução rítmica, enquanto o Contrabaixo e a Guitarra realizam a função harmônica. Estes, somados ao apoio rítmico da bateria, compõem o acompanhamento da música.

2.2 Softwares de acompanhamento

Um *software* de acompanhamento busca oferecer apoio harmônico e rítmico para que o usuário possa executar uma música. Normalmente, o utilizador da ferramenta toca uma melodia, mas dependendo da situação pode acompanhar a harmonia ou o ritmo gerado. Observando isso, alguns projetos começaram a surgir para atingir esses objetivos, e assim elaborar novas e diferentes formas para construção destas ferramentas (DAHIA et al., 2003).

Um desses métodos de implementação basea-se em modelos computacionais capazes de gerar quais notas serão utilizadas na composição dada uma certa entrada (JOHNSON-LAIRD, 1991). Geralmente a saída dessas aplicações são arquivos de áudio (*.mp3, .wav*), mas também podem ser gerados outros formatos que não reproduzem o som diretamente, mas armazenam as informações necessárias para a sua execução, como é o caso do MIDI (CASABONA; FREDERICK, 1988).

Outra forma é reutilizar fragmentos de gravações armazenadas em uma biblioteca, para que, conforme o usuário for inserindo os acordes, ritmos e outras informações os áudios adequados sejam acessados e reproduzidos (DAHIA et al., 2003).

2.3 Aplicações Web

O fundamento arquitetural da *World Wide Web* consiste em um modelo conhecido como cliente-servidor. Neste esquema o **cliente** realiza requisições para um **servidor** que deve retornar as informações solicitadas. Existem várias arquiteturas baseadas nesse modelo, sendo que nesta proposta será utilizado o conceito de *Single Page Application (SPA)*.

Tradicionalmente as aplicações *web* que envolvem interação com o servidor, geralmente, realizam o recarregamento total da página visível ao cliente. As *SPAs* apresentam como principal característica o fato de apenas atualizarem os elementos necessários da página, buscando manter-se o mais próximo da experiência de um *software desktop* ou *mobile*. Durante o desenvolvimento de uma SPA é recomendável que a aplicação seja decomposta em componentes individuais que possam ser substituídos ou atualizados independentemente. Neste modelo a interação com o servidor é feita por requisições assíncronas, normalmente utilizando o formato *JSON* e não mais utilizando apenas *HTMLs* (JADHAV; SAWANT; DESHMUKH, 2015).

Outra motivação para utilizar o modelo arquitetural *SPA* é que torna-se mais natural traduzir as boas práticas de engenharia de software aplicadas em outras áreas, principalmente no desenvolvimento de aplicações *desktop*, e adequá-las ao contexto da implementação de uma ferramenta *web*.

2.4 Computação em nuvem

Dentro da gama de publicações sobre a temática, o NIST *(National Institute of Standards and Technology - USA)* define a Computação em nuvem como um modelo que possibilita acesso sob demanda a recursos computacionais configuráveis, como, por exemplo armazenamento, rede, servidores e serviços, de forma rápida, com mínimo gerenciamento e pouca interação com o fornecedor dos serviços (MELL; GRANCE et al., 2011). O instituto também definiu as 5 principais características para o modelo de computação em nuvem:

- **Serviços sob demanda:** o cliente possui liberdade para alterar os recursos contratados, sendo esta efetuada automaticamente sem necessidade de interação humana.
- **Independência de localização:** os recursos devem ser disponibilizados utilizando redes, idealmente de alta velocidade, com intuito ser utilizado em plataformas heterogêneas, como, por exemplo celulares, tablets e computadores.
- **Repositório de recursos:** múltiplos usuários compartilham o mesmo recurso físico. É responsabilidade do provedor gerenciar e alocar adequadamente os recursos.
- **Elasticidade e Escalabilidade:** a elasticidade é a característica de prover e remover, em tempo de execução, os recursos. A escalabilidade define o aumento ou decréscimo da capacidade dos recursos do cliente.
- **Medição dos serviços:** os provedores monitoram e realizam medições do uso de cada recurso contratado, isto viabiliza a cobrança por uso, além de fornecer transparência.

Além disso, (MELL; GRANCE et al., 2011) apresenta um modelo conceitual para computação em nuvem, que se baseia em serviços e é dividido em camadas.

- ***Infrastructure-as-a-Service (IaaS)***: o usuário pode requisitar recursos de processamento, armazenamento, redes e outros aspectos fundamentais para computação. Neste ambiente o usuário é livre para executar e configurar quaisquer sistemas operacionais ou aplicações sem a necessidade de preocupação com o aspecto físico da infraestrutura.
- ***Platform-as-a-Service (PaaS)***: baseado na infraestrutura já ofertada pelo provedor o cliente apenas fica responsável por desenvolver sua ferramenta, sendo pra ele invisivel como o sistema operacional, redes e armazenamento estão configurados.
- ***Software-as-a-Service (SaaS)***: são aplicações, ou um conjunto delas, disponibilizados através da internet. Nessa camada apenas a aplicação é visível ao usuário sendo o restante responsabilidade do provedor do serviço

Com crescimento da computação em nuvem e do paradigma *"as a Service (aaS)"*, algumas novas classificações começaram a surgir em conjunto, assim como o *"Anything-as-a-Service (XaaS)"* (DUAN et al., 2015). Como o nome sugere, o **XaaS** traz a perspectiva que toda tecnologia disponibilizada por meio da internet pode ser considerada um novo conceito de serviço.

2.4.1 FaaS

Uma dessas novas abstrações é o modelo *"Function-as-a-Service (FaaS)"*, este pode ser entendido como um *PaaS*, em que a plataforma só ativa o código criado pelo usuário durante a execução de uma solicitação. No *FaaS*, o código do usuário é apenas uma função, dessa forma cada requisição é abstraída como uma chamada de função, porém localizada na nuvem. [(EYK et al., 2017). Tais funções devem ser, idealmente, pequenas, com responsabilidades únicas e sem necessidade de persistência de estado. Cada função só consome recursos do servidor no momento do processamento (KOLLER; WILLIAMS, 2017).

Combinando o benefício da arquitetura *Serverless*, de tornar responsabilidade do provedor os aspectos de gerenciamento, execução e medição do serviço, as características intrínsecas das *cloud functions* de tamanho e atribuição, obtenhe-se uma maior granularidade na possibilidade de pagamento do provedor (LYNN et al., 2017). Atualmente a metragem em milissegundos é a mais comum, isso possibilita que aplicações paguem apenas pelo uso real de processamento, evitando despesas com máquinas virtuais ociosas. Algumas plataformas existentes que ofertam esse serviço são: AWS Lambda, Google Cloud Functions, Fission e OpenWhisk.

As *Cloud functions* tendem a ser vantajosas em situações onde a demanada é intermitente ou com altas variações, pois não é interessante manter um serviço ativo de forma permanente para atender esse cenário.

2.4.2 Storage

Partindo do fato que em um ambiente de execução de um **FaaS** não há persistência de dados, existe a necessidade que as informações processadas pelas funções possam, de algum modo, serem armazenadas. Uma maneira de suprir essa lacuna é utilizar-se de *Cloud Storage*. Esta segue as mesmas características da computação em nuvem, ou seja, possibilita maior portabilidade dos dados, uma ilusão de armazenamento infinito e pagamento apenas por aquilo que se usa (WU et al., 2010). A maior vantagem de utilizar este tipo de armanzemento é que muitos provedores de serviços **(aaS)** também fornecem mecanismos para conexão entre ambos. Por exemplo, a Google Cloud Functions fornece mecanismos e documentação para conexão entre seus serviços.

3 Metodologia

Este capítulo dedica-se a especificar e explicar quais mecanismos e processos foram utilizados para o desenvolvimento da ferramenta.

3.1 Ferramentas e ambiente de desenvolvimento

Tecnologias *Web* fornecem maior flexibilidade e portabilidade para as aplicações. Quando utilizadas com princípios de responsividade e padrões de usabilidade permitem a utilização da plataforma em diversos dispositivos, sejam eles movéis ou não, com diferentes resoluções. Isso permite a disseminação da aplicação para seu público alvo (músicos e estudantes de música) que demandam, principalmente da praticidade.

Visando a manutenabilidade e evolução futura pela comunidade de software livre, este projeto adota desenvolvimento orientado a serviços, de forma a modulizar as tecnologias e torná-las especialistas. Normalmente, divide-se as responsabilidades do sistema em duas partes, uma focada em processamento dos dados da aplicação e a outra na interação com o usuário, são chamados de *Backend* e o *Frontend*, respectivamente. Neste projeto, ambas utilizam containers a fim de automatizar a configuração dos ambientes de desenvolvimento e produção, utilizando a aplicação **Docker**. O panorama geral das ferramentas que foram utilizadas é:

- **Docker:** Containerização (DOCKER, 2020)
- **Flask:** *microframework*, escrito em Python, para fornecer suporte ao desenvolvimento do *Backend* (FLASK, 2020)
- **MMA:** *software*, escrito em Python, responsável por gerar o acompanhamento musical (MMA, 2020)
- **Midifile:** biblioteca para leitura, escrita e sequenciamento de arquivos MIDI diretamente no browser (NFROIDURE, 2020)
- **ReactJS:** *framework* para construção de páginas utilizando Javascript (REACT, 2020)
- **Parcel:** empacotador de aplicações *web* (PARCEL, 2020)
- **WebAudioFont:** conjunto de recursos e tecnologias para reprodução de instrumentos musicais em um *browser*, incluindo uma biblioteca de timbres com diversos instrumentos virtuais (SURIKOV, 2020)

Vale destacar que algumas ferramentas neste projeto foram escolhidas através de um processo de filtragem, sendo esta realização essencial para promover melhor entendimento do problema e como estruturar a solução.

3.1.1 MMA - Music MIDI Accompaniment

Para chegar a escolha desta ferramenta o primeiro passo necessário foi elencar quais *softwares* estão atualmente disponíveis ao músico para geração de acompanhamento e que pudessem ser utilizados como base no desenvolvimento da solução. Com isso, foi realizado um levantamento, divido em duas tabelas, a primeira com *softwares* de acompanhamento, tabela 1 e a outra com escritores de partitura, tabela 2. A principal diferença entre as duas categorias é que essa última exige maior conhecimento do usuário sobre a transcrição de partituras, enquanto a primeira requer apenas conhecimento de cifragem básica. Ambas as listas contém informações sobre nome, licença e endereço do site oficial.

Comparando as informações da listagem e buscando mais informações sobre as possíveis candidatas, a ferramenta escolhida foi a **MMA - Music MIDI Accompaniment**, uma ferramenta de acompanhemento musical para ser utilizada na interface de linha de comando. Desenvolvida em Python, essa ferramenta basicamente converte um arquivo *'.mma'* para um arquivo *'.mid'* que pode ser executado por qualquer player de MIDI ou sequenciador externo.

Tabela 1 – Softwares de acompanhamento musical

Nome	**Licença**	**Site Oficial**
Band-In-a-Box	Proprietário	www.pgmusic.com/
ChordPulse	Proprietário	www.chordpulse.com/
iReal	Proprietário	irealpro.com/
MMA - Music MIDI Accompaniment	GPL	www.mellowood.ca/mma/

Tabela 2 – Softwares para escrita de partitura

Nome	**Licença**	**Site Oficial**
Aria Maestosa	GPLv2	ariamaestosa.github.io/ariamaestosa/docs
Capella	Proprietário	www.capella-software.com/
Denemo	GPL	www.denemo.org/
Dorico	Proprietário	new.steinberg.net/dorico/
Finale	Proprietário	www.finalemusic.com/
Forte	Proprietário	www.fortenotation.com/en/
Frescobaldi	GPLv2	frescobaldi.org/
Gregorio	GPLv3	gregorio-project.github.io/
Guitar Pro	Proprietário	www.guitar-pro.com/en/index.php

Continua na próxima página

Tabela 2 – continuação da tabela anterior

Nome	Licença	Fonte
Impro-Visor	GPLv2	www.cs.hmc.edu/ keller/jazz/improvisor/
LilyPond	GPLv3	lilypond.org/
Mozart	Proprietário	www.mozart.co.uk/
Mus2	Proprietário	www.mus2.com.tr/en/
MuseScore	GPLv2	musescore.com/
MusiCAD	Proprietário	musicad.com/introduction
Musink	Freeware	musink.net
MusiXTeX	GPLv2	ctan.org/pkg/musixtex?lang=en
Notation Composer	Proprietário	www.notation.com/
NoteFlight	Proprietário	www.noteflight.com/
NoteWorthy Composer	Proprietário	noteworthycomposer.com/
NOTION	Proprietário	www.presonus.com/products/notion
Overture	Proprietário	sonicscores.com/overture/
Philip's Music Writer	GPL	people.ds.cam.ac.uk/ph10/pmw.html
Rosegarden	GPL	rosegardenmusic.com/
ScoreCloud	Proprietário	scorecloud.com/
Sibelius	Proprietário	www.avid.com/sibelius
SmartScore	Proprietário	www.musitek.com/
TuxGuitar	LGPLv2	www.tuxguitar.com.ar/

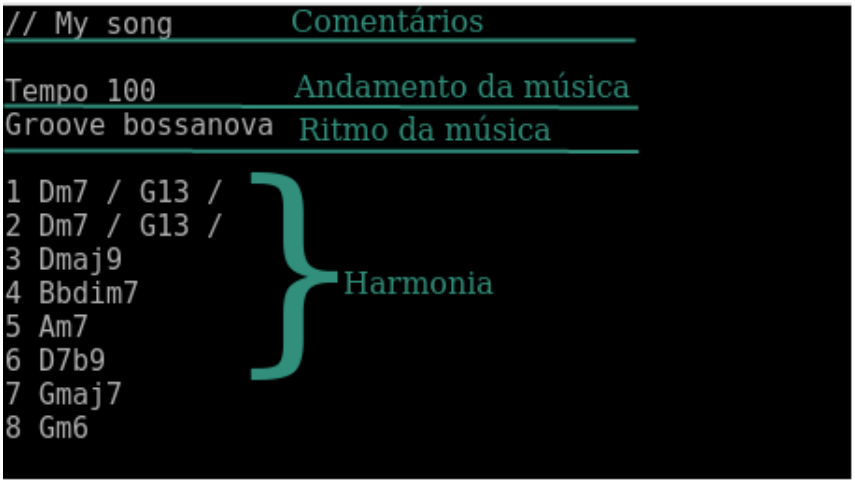


Figura 2 – Arquivo *'.mma'* para composição harmônica

Um arquivo *.mma*, mostrado na figura 2, deve conter no mínimo a especificação da harmonia, andamento (tempo) e o estilo musical de acompanhamento, denominado *Groove* na nomenclaruta do *software*, que baseado nessas informações gera um arquivo MIDI para a música.

O MMA contém alguns estilos musicais já estão catalogados, porém o usuário tem a autonomia para criação de novos ritmos através do próprio formato de arquivo *'.mma'*, como mostrado na figura 3. Este formato permite a definição da fórmula de compasso, definição de timbre e a escrita da célula do estilo, utilizando uma notação específica definida na documentação oficial

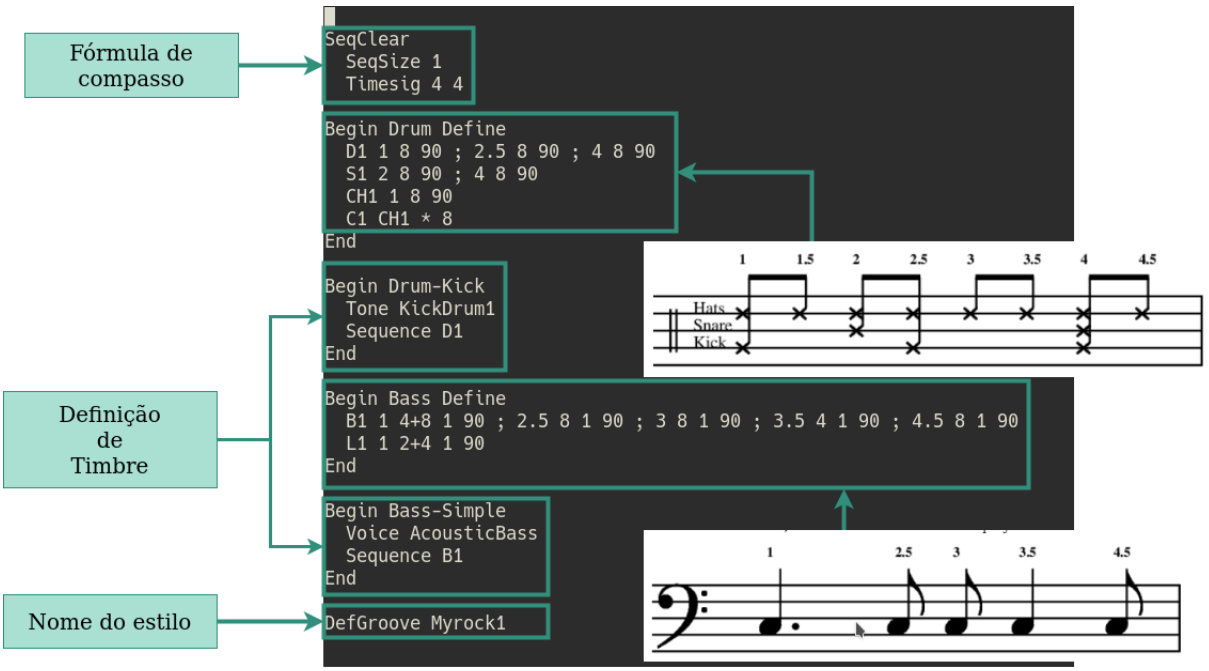


Figura 3 – Arquivo *'.mma'* para composição rítmica

3.1.2 Midifile e WebAudioFont

Sabendo que o MMA produz como saída arquivos MIDI, é necessário uma solução, compatível com navegadores, capaz de ler, sequenciar e reproduzir estes arquivos com fidelidade adequada de timbres. Para isso, identifcou-se duas bibliotecas que atendem os critérios necessários, o **Midifile** e o **WebAudioFont**

A primeira é responsável por ler e sequenciar o arquivo *.mid*. Já a segunda oferece uma biblioteca de timbres, e também implementa algumas funcionalidades extras, como troca em tempo de execução de quais instrumentos estão sendo tocados, equalizadores do áudio e possibilidade de inserção de novos *samples* para melhorar a qualidade sonora durante a composição. Não há dependência direta entre as duas opções escolhidas, no entanto, o **WebAudioFont** já apresenta exemplos da documentação com integração com a **Midifile**, de forma que aparentou natural utilizar as duas em conjunto.

3.2 Arquitetura

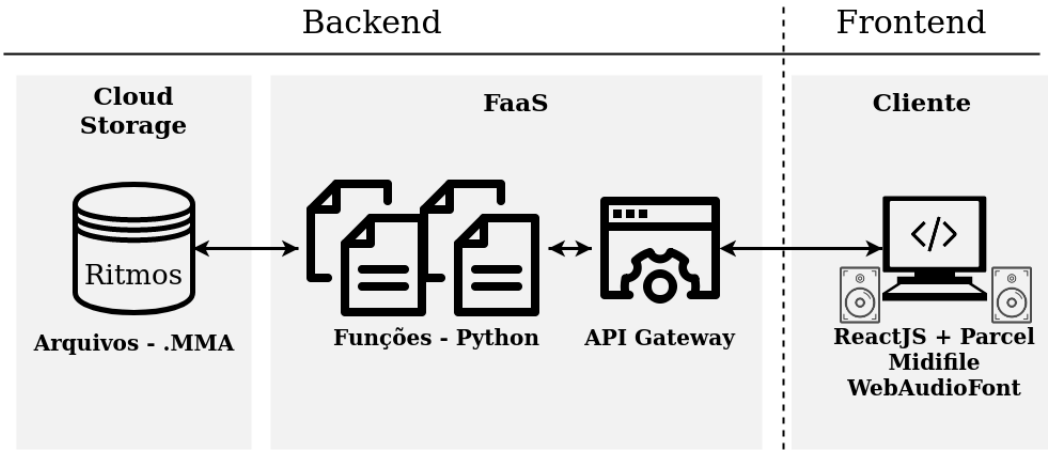


Figura 4 – Arquitetura da solução

Observando a visão arquitetural geral do projeto, figura 4, é possível visualizar a divisão entre o *Frontend*, executado no lado do cliente e responsável por todas atividades relacionadas a interação com o mesmo, desde a coleta dos dados, reprodução do aúdio, até outros controles da ferramenta. E o *Backend*, que depende das informações geradas pelo *Frontend*, sendo executado externamente e dedicando-se exclusivamente ao processamento das informações que retroalimentam a visão do usuário. Cada uma destas partes será detalhada nos próximos tópicos.

3.2.1 Backend

O *Backend*, como um todo, está projetado para utilizar os serviços de *Function-as-a-Service (FaaS)* e *Cloud Storage* viabilizados por provedores de computação em nuvem, no caso deste projeto a Google Cloud. Nessa arquitetura, a entrada dos dados é intermediada por uma *API Gateway* ofertada pelo próprio provedor. Este *Gateway* tem como objetivo realizar o controle das rotas e assim indicar qual função deverá ser executada, dependo da URL, como pode ser visto na figura 5.

As funções foram implementadas em *Python* sendo executadas no *framework Flask*, cada uma destas tem um objetivo único. A função **listgrooves()** apenas retorna o arquivo contendo a lista com os ritmos disponíveis. A rotina **createsong()** recebe como argumento um arquivo *'.json'*, vide um exemplo em A.1, contendo os elementos necessários para que, internamente, seja escrito um arquivo temporário *'.mma'*, exemplificado em A.2. Criado este arquivo, são invocadas as funções necessárias advindas do uso do *Music MIDI Accompaniment - MMA* como uma biblioteca de apoio. A saída desta função é um arquivo no formato *'.mid'*, sendo este o argumento retornado ao *Frontend*.

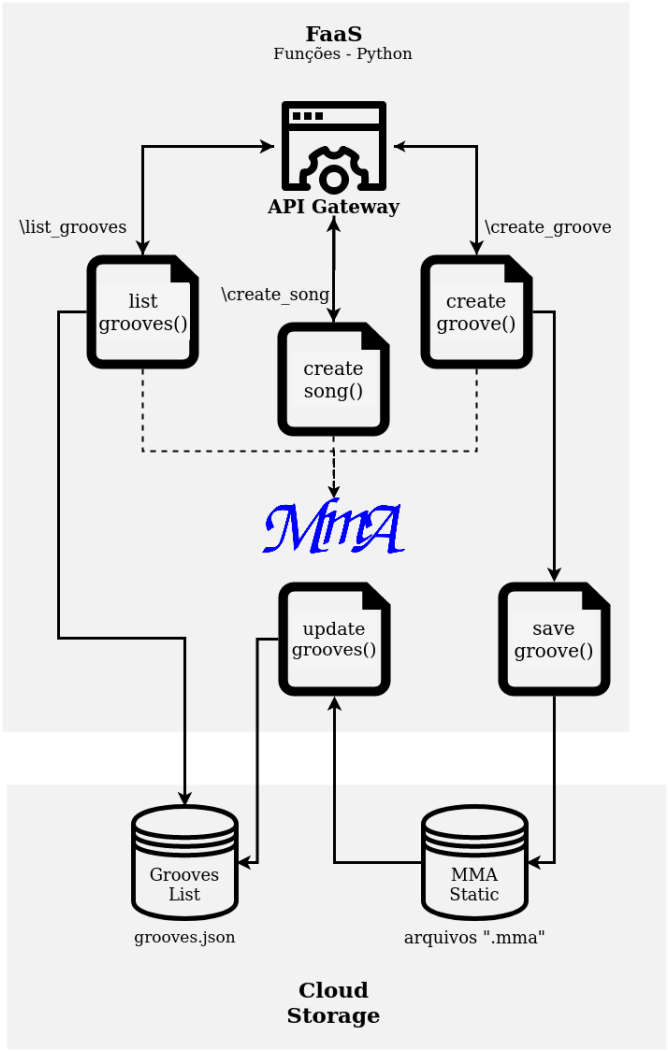


Figura 5 – Arquitetura da solução para *Backend*

A **creategroove()** é responsável por coletar um arquivo *'.mma'*, exemplo A.3, contendo a especificação necessária para criação de um novo ritmo. Esse arquivo é validado na função e, somente com resultado positivo, é invocada a próxima função, **savegroove()**. Esta outra realiza o *upload* do arquivo para o local adequado dentro da *Cloud Storage*. Feito isso, o provedor *Google Cloud* possibilita executar funções após alterações no *storage*, e com isso, sucede a execução da função **updategrooves()**. Esta atualiza o arquivo *grooves.json* contendo a listagem dos ritmos disponíveis na ferramenta. Em caso de falha em alguma das etapas deste sistema, a função é interrompida e o usuário é notificado sobre o ocorrido.

Para minimizar gastos durante a execução do desenvolvimento, o ambiente de teste e construção da solução do backend utilizou um container *Docker* com *Flask* para controle de rotas e execução das funções. As atividades relacionadas ao armazenamento de arquivos ocorreram utilizando o próprio sistema de arquivo local da máquina.

3.2.2 Frontend

Neste outro serviço, construído utilizando o empacotador *Parcel* combinado ao *framework* de Javascript *React*, o objetivo é disponibilizar as funcionalidades da aplicação por meio de uma interface gráfica. O Protótipo de alta fidelidade, figura 6, exemplifica como a área de criação, personalização e execução de músicas é visualizada pelo usuário da ferramenta. Além disso, traz a forma estrutural de implementação divida em três grandes componentes: *Header*, *Player* e *ChordChart*.

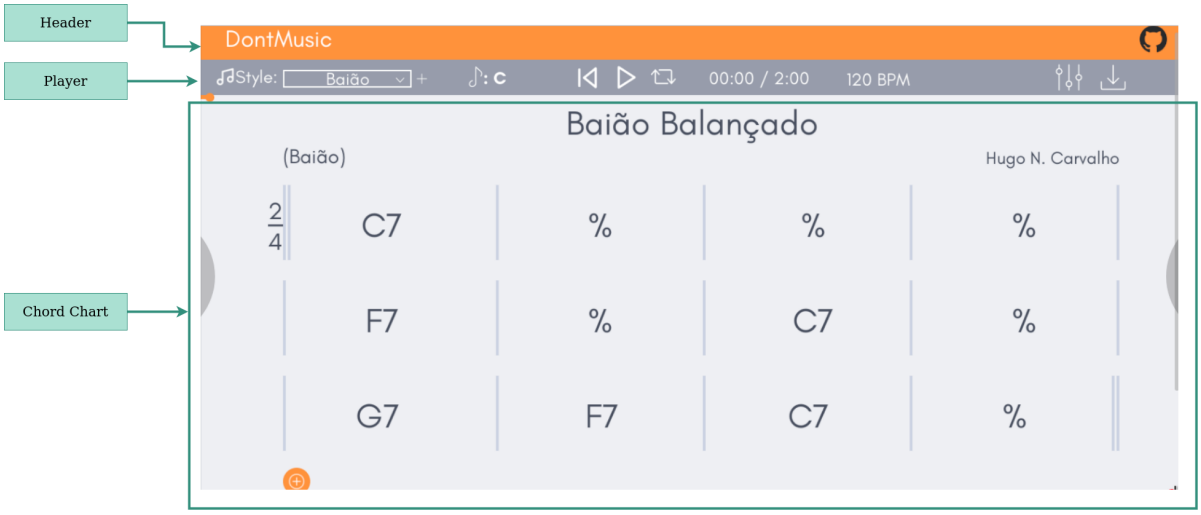


Figura 6 – Protótipo de alta fidelidade - Compositor de músicas

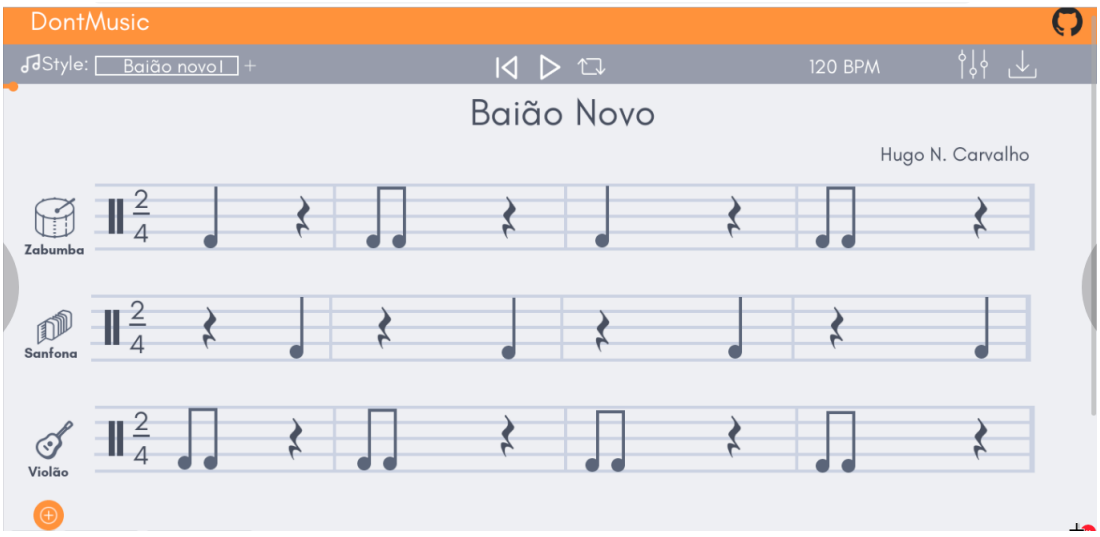


Figura 7 – Protótipo de alta fidelidade - Compositor rítmico

Combinando a figura 6 com a figura 8 têm-se uma melhor visualização da integração entre os componenentes internos do *Frontend*. O primeiro elemento traz apenas informações visuais, como, por exemplo o link deste projeto no Github. As funcionalidades

de controle e informações da execução está listadas elemento *Player*, sendo que este recebe o arquivo MIDI advindo da *API* externa, porém interfaciada pelo último componente.

O *ChordChart* é quem recebe e controla a inserção dos acordes, andamento (em BPMs) e qual ritmo foi selecionado. Estas informações são condensadas em um arquivo *.json* que é enviado ao *Backend* para a função **createsong()** através de requisão HTTP. Desta requisição, como já mencionado, a resposta é um arquivo *'.mid'* que é redirecionado ao componente responsável por executá-lo no próprio *browser* com apoio das bibliotecas *Midifile* (sequenciador) e *WebAudioFont* (timbres).

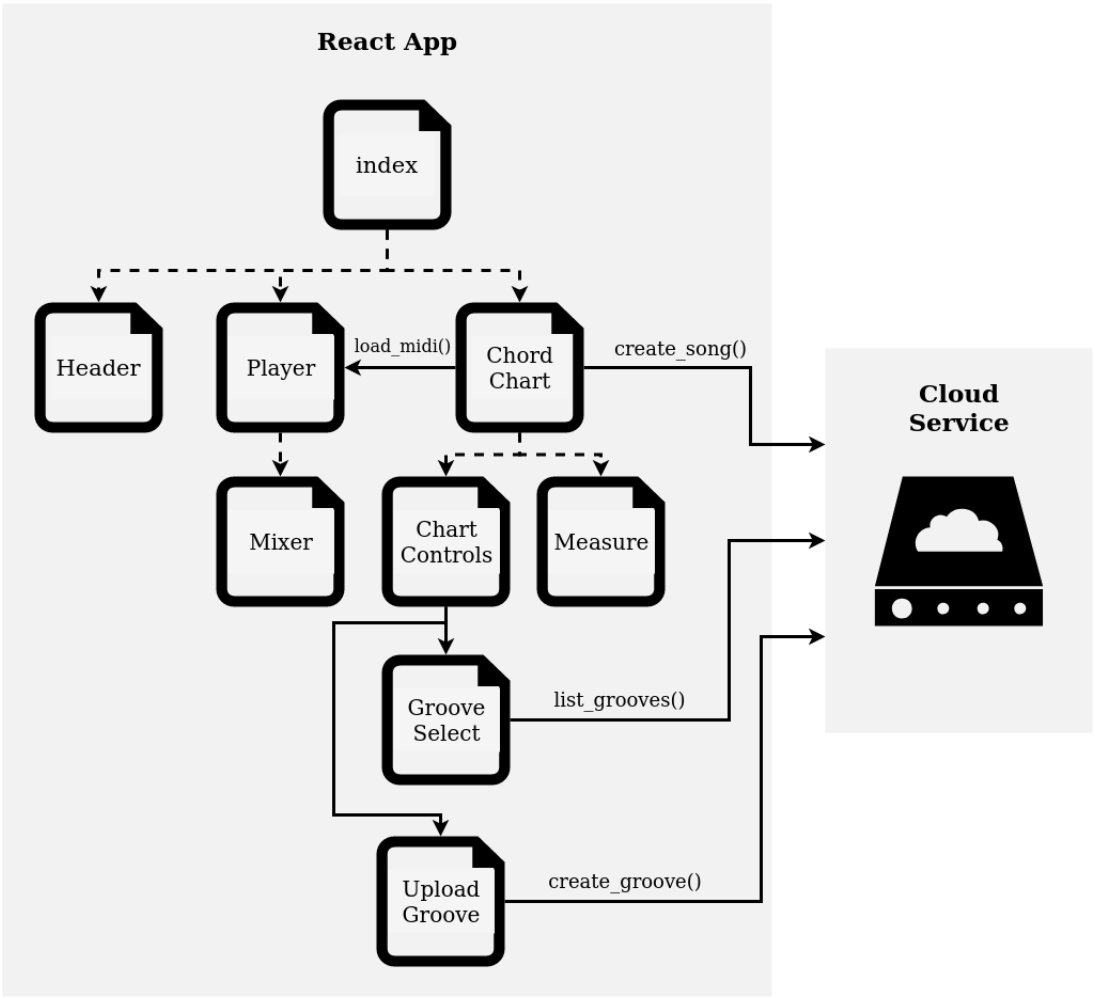


Figura 8 – Arquitetura da solução para *Frontend*

4 Resultados

Este capítulo mostra os principais resultados e produtos desenvolvidos durante a elaboração deste trabalho.

4.1 Prova de conceito

Após o sucesso da etapa anterior, inicializou-se a fase de implementações visando validar a arquitetura, ferramentas e fornecer uma base sólida para um desenvolvimento contínuo. O primeiro passo foi executar a correta configuração dos containers de cada serviço, bem como a comunicação entre estes e suas respectivas funções. Para avaliar a integração entre os dois serviços, foi criado um experimento básico em que o *Frontend* fornecia um arquivo válido .mma contendo uma música e deveria aguardar um arquivo .mid, o qual pudesse ser reproduzido através do sequenciador externo TiMidity++ (TIMIDITY, 2020).

Com este teste inicial bem-sucedido e reforçando a arquitetura proposta, o próximo passo foi estabelecer a execução do MIDI diretamente no browser. O diferencial neste aspecto foi contar com vários exemplos da própria documentação oficial do *WebAudioFont* (SURIKOV, 2020). Assim, estudando os códigos disponíveis, foi possível implementar o *player* e os controles essenciais do *software*. A partir dessa versão, que validou o fluxo completo, foi possível focar em outro pontos como listar os *grooves* disponíveis e a tratar da criação de novos ritmos. Já no *Frontend*, as atividades foram relacionadas a criação do design e integração interna dos componenentes, visando sempre uma boa experiência de uso da aplicação.

4.2 MMA - Pypi

Sabendo que a solução proposta utiliza como base fundamental o *software MMA - Music MIDI Accompaniment*, a primeira etapa foi analisar e entender de quais formas este poderia ser utilizado como auxiliar do desenvolvimento. O MMA é distribuido através de um arquivo compactado no formato *.tar.gz*, não tendo integração com o PyPi(PYPI, 2020), o serviço oficial de disponibilização de pacotes em Python.

Este tipo de estratégia de distribuição torna o uso do MMA em outras aplicações Python um pouco inconveniente. Sendo assim, foi necessário fazer uma refatoração no código pra permitir a disponibilização no repositório oficial do PyPi. Com isso tornou-se possível tanto executar diretamente na máquina (linha de comando) ou consumir o

MMA como biblioteca externa. Outro benéficio é a facilidade na distribuição e instalação, ideais para utilização em uma aplicação em nuvem. Este processo de empacotamento foi realizado no contexto deste trabalho e está disponibilizado através deste link. Entrei em contato com o desenvolvedor para comunicar o empacotamento da ferramenta e tive um *feedback* positivo pela iniciativa.

4.3 Compositor de músicas

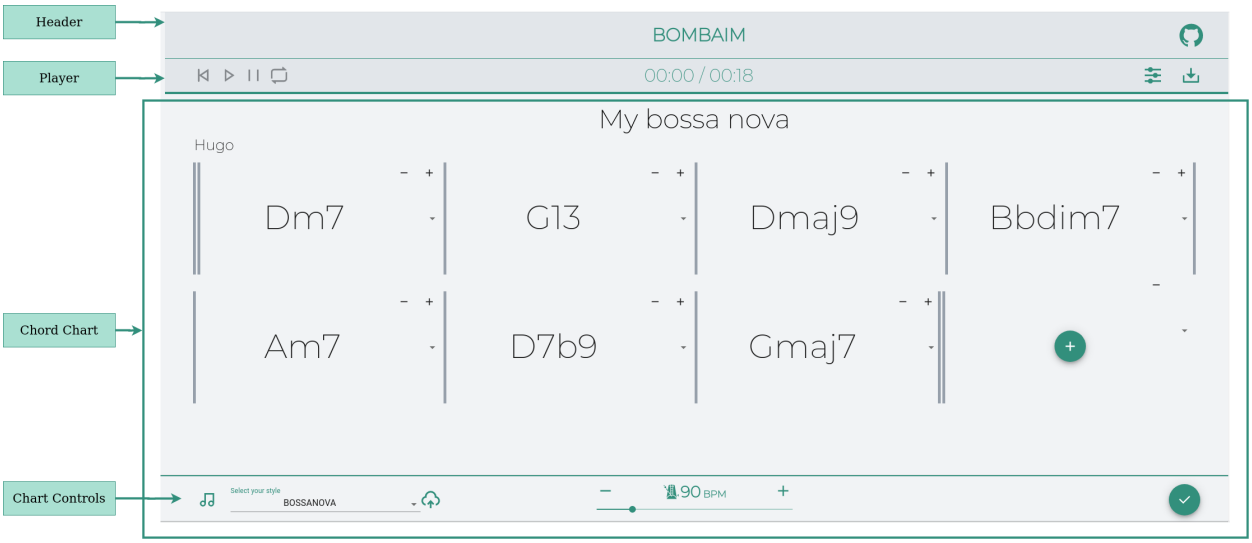


Figura 9 – Tela do Compositor de músicas

Com o sucesso na fase de prova de conceito e no empacotamento do MMA, iniciou-se a construção da tela do compositor de música, a principal e única desta aplicação. Comparando a figura 9 com o protótipo de alta fidelidade da figura 6, nota-se que o padrão da divisão entre os componentes foi mantido, porém ligeiramente alterado em alguns aspectos. O novo design tenta priorizar a clareza na localização das funcionalidades e facilitar o uso das mesmas, como é o proposto por outras ferramentas citadas neste trabalho. Dessa forma, alguns componentes são apenas para visualização de dados, como, por exemplo o *Header*, que apenas expõe o nome da aplicação e um link para o repositório do código no Github.

4.3.1 Player

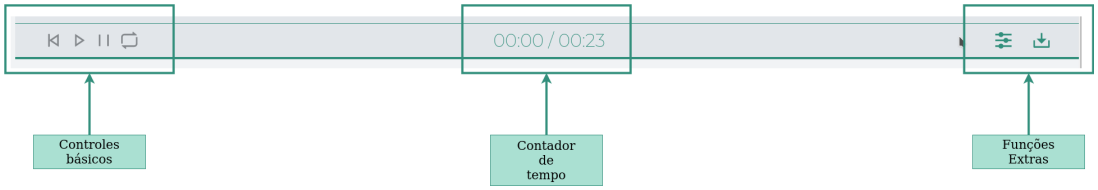


Figura 10 – Visão ampliada e detalhada do *player*

O objetivo deste elemento, figura 10, na aplicação é oferecer ao usuário ferramentas para controle e customização da reprodução do áudio do acompanhamento musical, e está dividido em três partes: controles básicos, contador de tempo e funções extras. Os controles básicos estão dispostos através de símbolos já consolidados nos tocadores de aúdio (RIBEIRO et al., 2005), são eles: ir para o início (*skip back*), tocar (*play*), pausar (*pause*) e repetir (*repeat*), na ordem da esquerda para a direita da imagem 11.

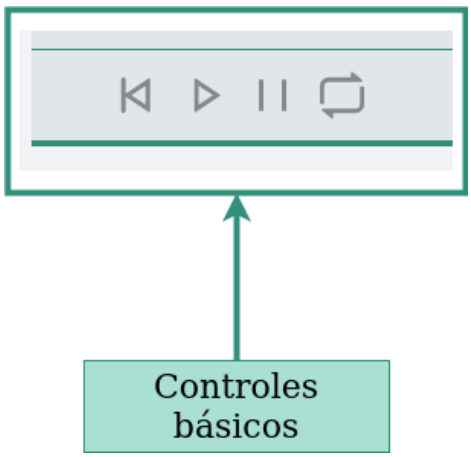


Figura 11 – Visão ampliada e detalhada dos controles básicos do *player*

Estas opções, por sua vez, invocam funções especializadas, cada uma responsável por controlar diretamente o contexto de áudio do browser e assim provocar o resultado esperado. Um exemplo desse fluxo ocorre na função executada ao pressionar o botão **play**, que acessa o contexto de áudio e aguarda o sucesso dessa operação para então iniciar a reprodução do áudio. Não sendo respeitada essa sequência, resultados inesperados podem acontecer devido a assincronia do Javascript.

O segundo elemento que compõem o componente *Player* é um contador visual do tempo. Apresenta uma contagem numérica formatada em **Minutos:Segundos** incluindo o **tempo atual de execução / tempo total de duração** e uma barra inferior que se preenche de outra cor para demonstrar o progresso da execução da música, como mostrado na figura 12.

Figura 12 – Visão ampliada do contador de tempo e barra de progresso do *player*

Ambas as contagens baseam em tempos fornecidos pelo arquivo MIDI gerado no *Backend*. Sendo que o tempo de execução é alterado a cada novo passo na função **tick()**. O código 12 exemplifica como os dois elementos são calculados e alterados além da chamada pela função **tick()**.

```
function tick(){
    // Atualiza o timer numérico e a barra de progresso
    this.updateTimer(song.currentSongTime, song.song.duration);

    //... atividades do sequenciador e leitor do arquivo MIDI

    // chamada recursiva vinculada ao frame rate da janela.
    window.requestAnimationFrame(function (t) {
        tick();
    });
}
```

Listing 4.1 – Trecho de código das funções do contador de tempo do *Player*

O último bloco é o das funções extras, figura 13, que inclui a possibilidade de download do arquivo *.mid* e ativação do Mixer descrito na seção 4.3.2.

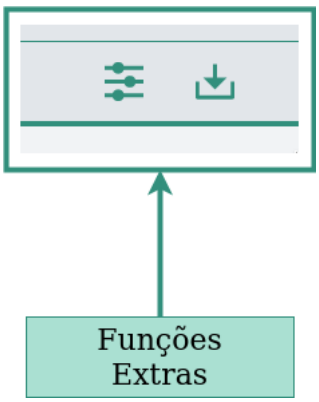


Figura 13 – Visão ampliada e detalhada dos controles extras do *player*

4.3.2 Mixer

A outra função extra do *Player* é um *mixer*, ícone à esquerda na figura 13, que permite alterar o instrumento e o volume de cada faixa do arquivo que será executado. Ao

clicar nessa opção, é apresentada ao usuário, figura 14, a janela para alterar estas informações de forma rápida e sugestiva. Agregar essa funcionalidade na ferramenta possibilita oferecer maior reflixibilidade e customização para o usuário final.

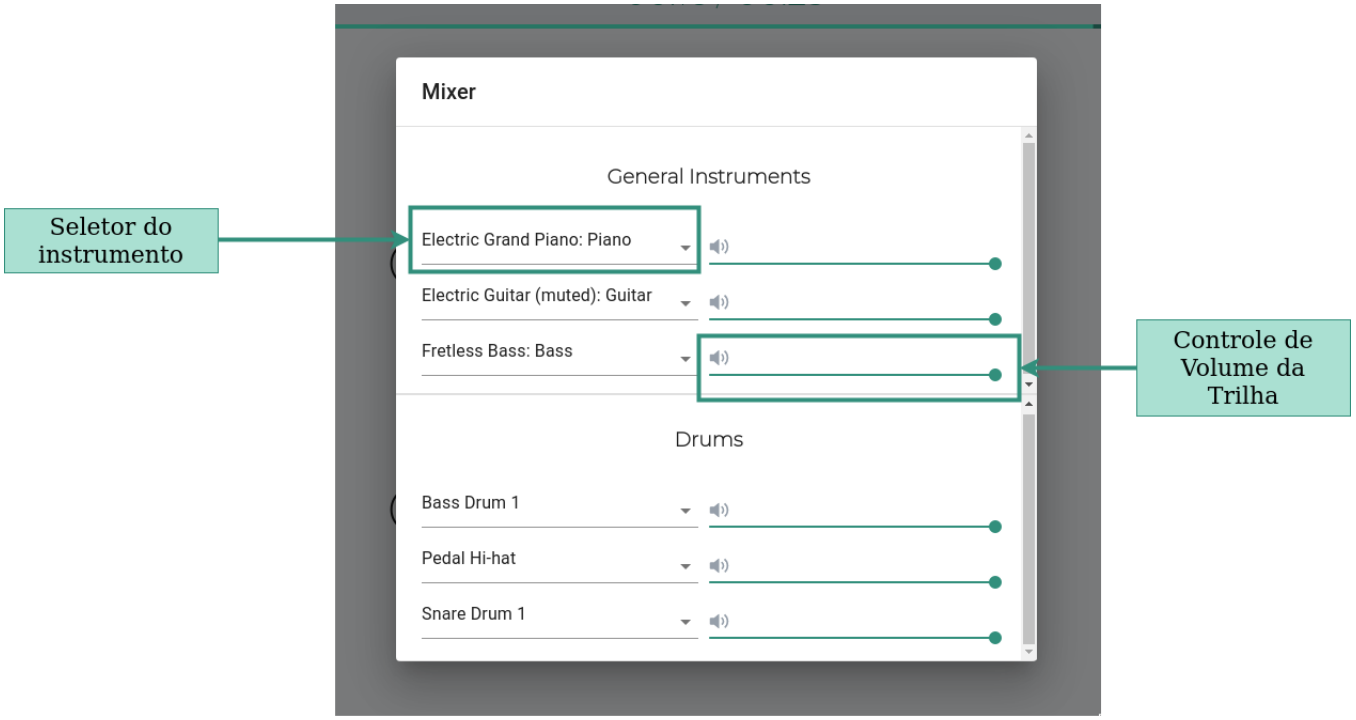


Figura 14 – Visão do *mixer* para controle das faixas do acompanhamento

A implementação deste controle foi viável, pois a própria biblioteca *WebAudioFont* oferece os mecanismos necessários para detectação de quais são os instrumentos requisitados pelo arquivo MIDI e atribuição, em tempo de execução, de um novo timbre para a linha. Observando o código 4.2, o nome do instrumento e o volume são argumentos na função **queueWaveTable()**. Essa possilidade de alteração do acompanhamento, aliada a numerosa quantidade de timbres já oferecidos, juntamente com perspectiva oferecida pela documentação para criação de novos *samples* (SURIKOV, 2020) fortalece a ideia desta ser uma ferramenta de crescimento colaborativo.

```
1
2 function sendNotes(song, songStart, start, end, audioContext, input,
    player){
3       var instrument = track.info.variable;
4       var volume = track.volume / 7;
5
6       player.queueWaveTable(audioContext, input, instrument, when,
    beat.n, duration, volume);
7    }
```

Listing 4.2 – Trecho de código da função para execução dos instrumentos

4.3.3 ChordChart

O mais importante componente da página de composição de músicas é o *Chord Chart*, que traduzido livremente significa o mesmo que a cifra da música. No entanto, como a figura 9 mostra, não trata apenas da inserção dos acordes, mas também de coletar o andamento (tempo) e seleção de qual ritmo a harmonia deve seguir, ambos aqui considerados *Chart Controls*.

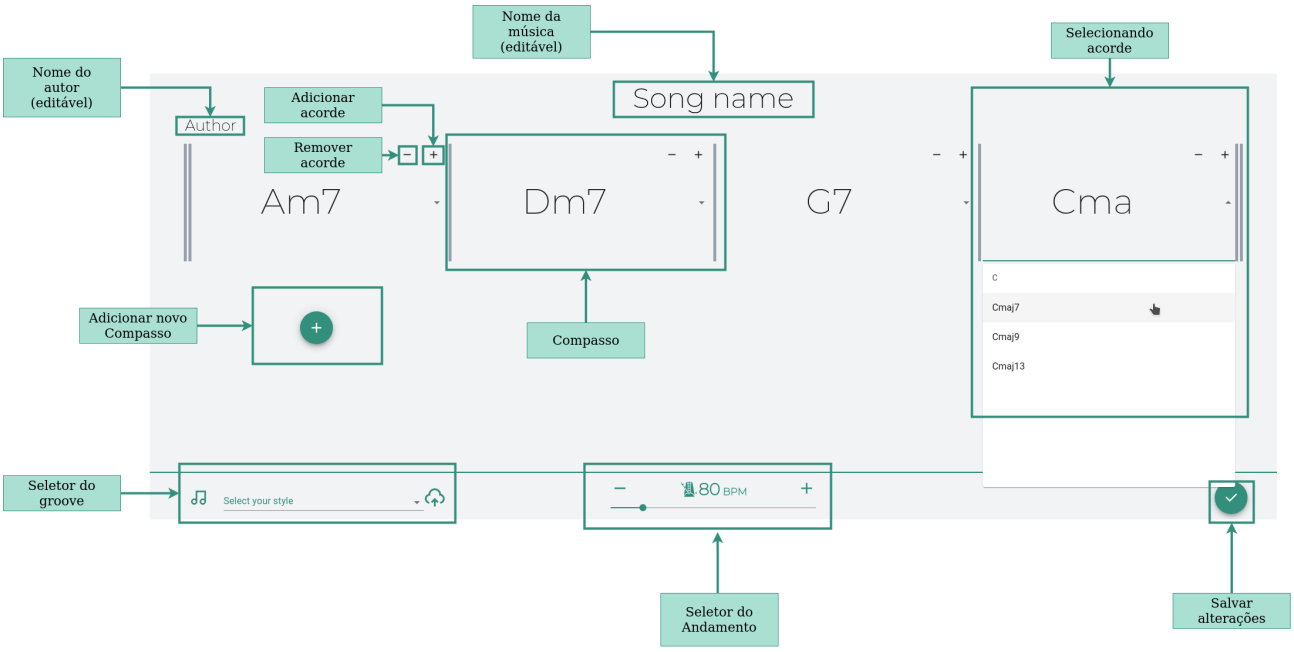


Figura 15 – Visão detalhada das funcionalidades presentes no *ChordChart*

O registro da sequência harmônica, figura 15, é feita por compasso e, havendo necessidade pode-se criar outro através do botão indicado. Um compasso pode conter um ou mais acordes, sendo que a inserção é acionada através do botão no canto superior do compasso indicado pelo símbolo +. O ícone ao lado, representado com -, realiza a exclusão do acorde do seu respectivo compasso, caso seja retirado o único acorde do compasso este é excluído totalmente.

O MMA entende que cada acorde em um compasso é uma semínima, e irá sempre preencher o compasso por igual. Assim como na figura 16, temos, por exemplo que uma fórmula de compasso 4/4 se apenas um acorde for fornecido este executará os 4 tempos, caso dois acordes sejam escritos o resultado será o primeiro acorde nos tempos 1 e 2 e outro nos tempos 3 e 4 e assim sucessivamente. O MMA é capaz de lidar com tempos compostos, mas é necessário atenção, pois por sempre se tratar de semínimas a escrita de um tempo 5/8 não deve ser anotada como 5, mas sim 2.5.

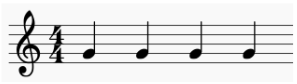
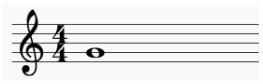
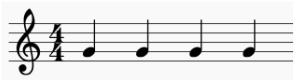
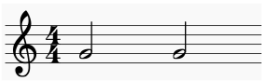
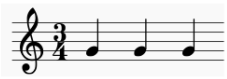
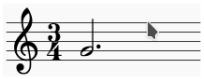
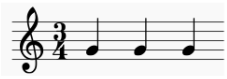
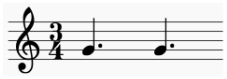
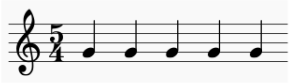
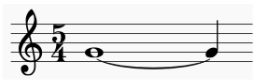
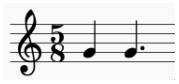
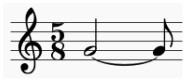
Fórmula de compasso	Fórmula de compasso MMA	Resultado MMA	Quantos acordes inseridos?	MMA executa
4/4	4		1	
4/4	4		2	
3/4	3		1	
3/4	3		2	
5/4	5		1	
5/8	2.5		1	

Figura 16 – Tabela exemplificando fórmula de compasso sob a visão do MMA

Sabendo dessas regras para inserção, basta preencher os espaços destinados, conforme destacado na imagem 15 com os acordes desejados. Utilizando-se de uma combo box personalizada, cada campo contém todos acordes válidos para o MMA, porém existe a possibilidade de buscar o acorde e só então selecioná-lo. Todas as opções foram geradas através do algoritmo 4.3, sendo o reseultado armazenado em um arquivo *.json*, desta forma, o acesso as opções é imediato e não necessidade de requisição para carregamento.

```
1 from mma.MMA import chordtable
2
3 chordlist = chordtable.chordlist
4 notes = ["C", "C#", "Db", "D", "D#", "Eb", "E", "E#", "Fb", "F", "F#", "
    Gb", "G", "G#", "Ab", "A", "A#", "Bb", "B", "B#", "Cb"]
5
6 json = {note: [note + ext for ext in chordlist]
7         for note in notes}
```

Listing 4.3 – Algoritmo para gerar arquivo *chords.json*

4.3.4 Chart Controls

A escolha do ritmo é um dos poucos elementos que executa requisão externa a aplicação do *Frontend*, essa ocorre assim que o componente é montado pelo *framework* executando a chamada das funções internas e, por consequência, ativando a comunicação

com o *Backend*. A listagem atualizada é coletada e tratada até devolver ao usuário em forma de *combo box*, conforme mostra a figura 17. É importante lembrar que há sempre a necessidade da selação de um ritmo, pois não existe um valor predeterminado.

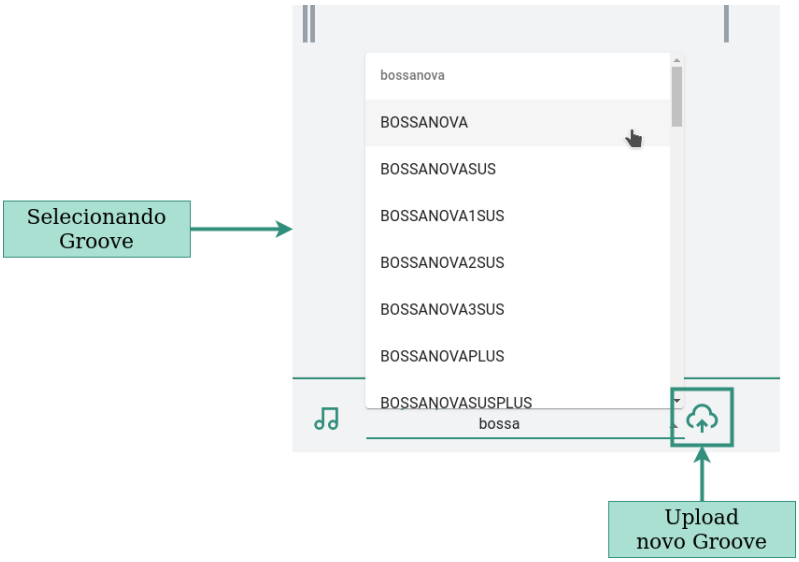


Figura 17 – Visão ampliada do seletor de ritmos

O seletor do andamento da música é bastante simples na sua utilização e também na sua implementação. Podendo ser acionado de duas formas, deslizando a barra inferior ou através dos ícones de - e +, ambos indicados na figura 18, o tempo é alterado automamticamento no *json* que será enviado para gerar o acompanhamento.

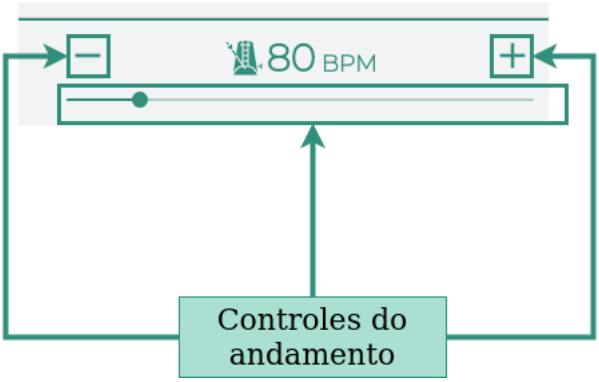


Figura 18 – Visão ampliada do seletor de andamento e seus controles

4.3.5 Upload Groove

O objetivo inicial da aplicação era construir um editor de *grooves*, como visto no protótipo na figura 7, porém devido a limitações de tempo não foi possível desenvolver essa funcionaliade e optou-se por incluir somente uma caixa de diálogo para que o usuário possa subir um groove já criado no formato MMA, como mostrado na figura 17.

Quando clicado no ícone, a aplicação apresenta uma *modal* com instruções e links para o material de apoio para criação de novos *grooves*. Todos os ritmos adicionados são compartilhados com todos usuários da plataforma, gerando assim um ambiente colaborativo. Para controlar e manter a integridade dos ritmos, é necessário que o nome do arquivo a ser inserido ainda não esteja cadastrado. Isso possibilita que não haja sobrescrita nos ritmos e que não sejam aceitas modificações.

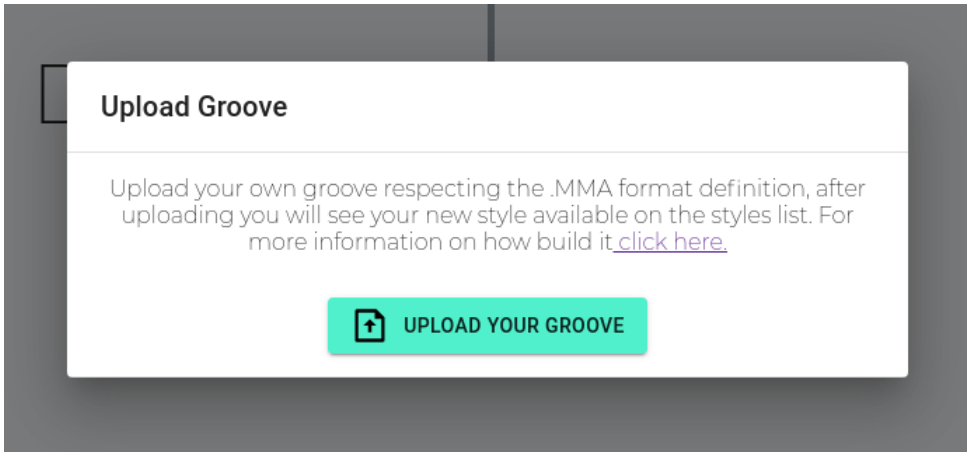


Figura 19 – Visão ampliada do upload de novo groove

5 Conclusão

Este trabalho mostra o desenvolvimento de uma ferramenta para acompanhamento musical na *web*. A aplicação está disponibilizada na forma de *software* livre, portanto consiste numa contribuição para todos os usuários que possam se interessar nesse tipo de funcionalidade de forma gratuita e facilmente disponível. Esta solução, nomeada Bombaim, buscou oferecer os músicos, em especial os estudantes de música, uma nova ferramenta preocupada em auxiliar as suas atividades diárias, focada na praticidade e oferecendo customização, ainda limitada, mas que pode render bons resultados futuros.

As tecnologias adotadas mostraram-se adequadas para a implementação dessa aplicação, respeitando o padrão desenhando na arquitetura e atingindo os objetivos esperados. O *MMA - Musical MIDI Accompaniment*, após o empacotamento como uma biblioteca mostrou-se capaz de desempenhar o papel de gerador dos acompanhamentos dentro da aplicação. As ferramentas usadas no *Frontend* também mostraram-se robustas, principalmente o *WebAudioFont* que foi utilizado como biblioteca de instrumentos virtuais.

5.1 Trabalhos futuros

Um objetivo de desenvolvimento imediato dessa aplicação é a implementação do editor para criação de novos ritmos, como descrito no protótipo, figura 7. Não foi possível desenvolver o editor no tempo alocado para o projeto, principalmente pela dificuldade em apresentar a linguagem do MMA de uma forma mais visual e amigável. Na mesma linha, seria interessante criar uma forma mais intuitiva para inserção de acordes. A representação de acordes é feita de maneira diferente em diferente lugares no mundo, então uma representação visual pode ser mais adequada do ponto de vista da internacionalização da aplicação.

Além disso, atualmente, a aplicação só encontra-se estável no Google Chrome e derivados. Tentativas de execução da aplicação foram realizadas juntos ao Mozilla Firefox, porém a qualidade da reprodução áudio fica abaixo do esperado, ou aparenta estar fora de sincronia. A origem deste comportamento ainda não foi encontrada e precisa de investigação.

Referências

BAND-IN-A-BOX. 2019. Disponível em: <https://www.pgmusic.com/>. Citado na página 23.

CASABONA, H.; FREDERICK, D. *What is MIDI?* [S.l.]: Alfred Music, 1988. Citado na página 26.

DAHIA, M. et al. Generating rhythmic accompaniment for guitar: the cyber-joão case study. In: SN. *Proceedings IX Brazilian Symposium on Computer Music*. [S.l.], 2003. Citado na página 26.

DOCKER. *Enterprise Container Platform | Docker*. 2020. <https://www.docker.com/>. (Accessed on 12/08/2020). Citado na página 29.

DUAN, Y. et al. Everything as a service (xaas) on the cloud: origins, current and future trends. In: IEEE. *2015 IEEE 8th International Conference on Cloud Computing*. [S.l.], 2015. p. 621–628. Citado na página 28.

EYK, E. V. et al. The spec cloud group's research vision on faas and serverless architectures. In: ACM. *Proceedings of the 2nd International Workshop on Serverless Computing*. [S.l.], 2017. p. 1–4. Citado na página 28.

FLASK. *Flask | The Pallets Projects*. 2020. <https://www.palletsprojects.com/p/flask/>. (Accessed on 12/08/2020). Citado na página 29.

IREAL. 2019. Disponível em: <https://irealpro.com/>. Citado na página 23.

JADHAV, M. A.; SAWANT, B. R.; DESHMUKH, A. Single page application using angularjs. *International Journal of Computer Science and Information Technologies*, Citeseer, v. 6, n. 3, p. 2876–2879, 2015. Citado na página 26.

JOHNSON-LAIRD, P. N. Jazz improvisation: A theory at the computational level. In: . [S.l.: s.n.], 1991. Citado na página 26.

KOLLER, R.; WILLIAMS, D. Will serverless end the dominance of linux in the cloud? In: ACM. *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. [S.l.], 2017. p. 169–173. Citado na página 28.

LYNN, T. et al. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In: IEEE. *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.], 2017. p. 162–169. Citado na página 28.

MED, B. *Teoria da música*. [S.l.]: Brasília: Musimed, 1996. v. 996. Citado na página 25.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National , 2011. Citado na página 27.

MMA. 2020. Disponível em: <https://www.mellowood.ca/mma/index.html>. Citado 2 vezes nas páginas 23 e 29.

MUSESCORE. 2019. Disponível em: <https://musescore.com>. Citado na página 23.

NFROIDURE. *nfroidure / midifile*. 2020. <https://github.com/nfroidure/midifile>. (Accessed on 12/08/2020). Citado na página 29.

PARCEL. *Parcel*. 2020. <https://parceljs.org/>. (Accessed on 12/08/2020). Citado na página 29.

PYPI. *Find, install and publish Python packages with the Python Package Index*. 2020. <https://pypi.org/>. (Accessed on 12/08/2020). Citado na página 37.

REACT. *A JavaScript library for building user interfaces*. 2020. <https://reactjs.org/>. (Accessed on 12/08/2020). Citado na página 29.

RIBEIRO, B. C. et al. *Oficina de Rdio Recursos de Áudio na WEB*. 2005. <http://www.usp.br/nce/midiasnaeducacao/pdfs/CA_Oficina_Radio.pdf>. Citado na página 39.

SURIKOV. *WebAudioFont / webaudiofont*. 2020. <https://surikov.github.io/webaudiofont/>. (Accessed on 12/08/2020). Citado 3 vezes nas páginas 29, 37 e 41.

THÉBERGE, P. *Any sound you can imagine: Making music/consuming technology*. [S.l.]: Wesleyan University Press, 1997. Citado na página 23.

TIMIDITY. *Timidity++*. 2020. <http://timidity.sourceforge.net/>. (Accessed on 12/08/2020). Citado na página 37.

WU, J. et al. Cloud storage as the infrastructure of cloud computing. In: IEEE. *2010 International Conference on Intelligent Computing and Cognitive Informatics*. [S.l.], 2010. p. 380–383. Citado na página 28.

Apêndices

APÊNDICE A – Exemplos de arquivos

```
{
    "name":"My bossa nova",
    "author":"Hugo",
    "tempo":"80",
    "groove":"BOSSANOVA",
    "measures":
        {
            "1":["Dm7"],
            "2":["G13"],
            "3":["Dmaj9"],
            "4":["Bbdim7"],
            "5":["Am7"],
            "6":["D7b9"],
            "7":["Gmaj7"]
        }
}
```

Listing A.1 – Exemplo de JSON enviado pelo *Frontend*

```
// Name Song name
// Author Author
Lyric On
Tempo 80
Groove BOSSANOVA
1 Dm7 [m1c1 ]
2 G13 [m2c1 ]
3 Dmaj9 [m3c1 ]
4 Bbdim7 [m4c1 ]
5 Am7 [m5c1 ]
6 D7b9 [m6c1 ]
7 Gmaj7 [m7c1 ]
```

Listing A.2 – Exemplo de arquivo .mma gerado no *Backend* para criação música

```
SeqClear
SeqSize 1
Timesig 4 4

Begin Drum Define
  D1 1 8 90 ; 2.5 8 90 ; 4 8 90
  S1 2 8 90 ; 4 8 90
  CH1 1 8 90
  C1 CH1 * 8
End
```

```
11
12  Begin Bass Define
13    B1 1 4+8 1 90 ; 2.5 8 1 90 ; 3 8 1 90 ; 3.5 4 1 90 ; 4.5 8 1 90
14    L1 1 2+4 1 90
15  End
16
17  Begin Chord Define
18    C1 1 2+4 80; 4 4 80
19  End
20
21  Begin Drum-Kick
22    Tone KickDrum1
23    Sequence D1
24  End
25
26  Begin Drum-Snare
27    Tone SnareDrum1
28    Sequence S1
29  End
30
31  Begin Drum-HH
32    Tone ClosedHiHat
33    Sequence C1
34  End
35
36  Begin Bass-Simple
37    Voice AcousticBass
38    Sequence B1
39  End
40
41  Begin Bass-LeftHandPiano
42    Voice Piano1
43    Sequence L1
44    Octave 3 // This a new command, but simple to understand
45  End
46
47  Begin Chord-RightHandPiano
48    Voice Piano1
49    Sequence C1
50  End
51
52  DefGroove Testgroove
```

Listing A.3 – Exemplo de arquivo .mma para criação de novo estilo