YASQ: Uma plataforma de salas virtuais colaborativas para reprodução de músicas em tempo real[1]

Arthur Bassotto Ziero[2], Felipe Martin Sampaio[3]

Instituto Federal do Rio Grande do Sul - *Campus* Farroupilha
Tecnologia em Análise e Desenvolvimento de Sistemas
95174-274 – Farroupilha – RS – Brasil

arthurbziero@gmail.com, felipe.sampaio@farroupilha.ifrs.edu.br

Resumo. *A música tem papel fundamental no entretenimento individual, mas também é de extrema importância social, unindo pessoas e fortalecendo suas conexões. Ao explorar essa dimensão social da música, foi identificada a necessidade de soluções que permitam que grupos de amigos apreciem a música de forma sincronizada e interativa. Este trabalho propõe a criação de uma aplicação web composta por um frontend e backend, que oferece uma sala virtual onde os participantes podem reproduzir faixas de áudio em sincronia e interagir por meio de um chat. Essa solução visa promover uma experiência musical compartilhada entre grupos de amigos.*

Abstract. *Music plays a fundamental role in individual entertainment, but it is also of great social importance, bringing people together and strengthening their connections. By exploring this social dimension of music, the need for solutions that allow groups of friends to enjoy music in a synchronized and interactive way has been identified. This work proposes the creation of a web application consisting of a frontend and backend, which provides a virtual room where participants can play audio tracks in sync and interact through a chat. This solution aims to promote a shared musical experience among groups of friends.*

1. Introdução

A música é uma arte que combina elementos sonoros, como ritmo, melodia, harmonia e letras (quando presentes), para criar experiências auditivas. Através dessa expressão artística é possível evocar emoções, transmitir histórias e conectar pessoas. Ao passo que evidências científicas acumulam-se, demonstrando que ouvir música possui efeitos benéficos nas emoções, na redução do estresse e na percepção da dor, o uso da música em tratamentos terapêuticos e para melhoria da saúde têm se tornado cada vez mais aceitos (HANSER, 2010).

Além dos benefícios emocionais e mentais de se ouvir música, essa também desempenha um papel significativo na sociedade, proporcionando um meio de comunicação emocional e conectando indivíduos. A música tem o poder de promover um senso de pertencimento a grupos sociais, permitindo que as pessoas se sintam mais

[1] Artigo científico referente ao Trabalho de Conclusão de Curso do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal do Rio Grande do Sul - *Campus* Farroupilha.
[2] Aluno matriculado no Trabalho de Conclusão de Curso.
[3] Professor orientador do Trabalho de Conclusão de Curso.

próximas de seus amigos, expressem sua identidade e valores aos outros, e melhorem sua compreensão dos ambientes sociais (SCHÄFER, T. et al., 2013, p. 6).

Com a popularização do acesso à Internet, a forma como se consome música vem se modificando significativamente, mudando de rádios, CDs e MP3 *players* para serviços on-line que são única e exclusivamente dedicados à reprodução dessas mídias de maneira digital. Desta maneira, uma ação que era muitas vezes física, requerendo diferentes dispositivos e mídias para que se pudesse escutar música, têm migrado para serviços de assinatura de *streaming de* áudio e/ou vídeo, como por exemplo, YouTube[4], Spotify[5], SoundCloud[6], e diversos outros.

Estas são aplicações com foco na reprodução de músicas de maneira individual e que dado este problema, o resolvem de maneira muito efetiva. Porém, tratando-se de escutar música em um grupo de pessoas, de maneira sincronizada e interativa, as plataformas apresentadas não atendem de forma satisfatória este requisito. O Spotify permite reprodução de músicas sincronizadas mas não permite interatividade, e o YouTube permite que usuários conversem, mas não permite que estes grupos controlem em tempo real a música que escutam, não havendo unanimidade neste quesito.

Tendo em conta o problema demonstrado, este artigo apresenta o desenvolvimento de uma uma solução com foco na experiência de escutar música em um grupo de pessoas, para que possam compartilhar momentos juntos e também seus gostos musicais através da execução sincronizada de faixas de áudio. A solução proposta baseia-se em uma aplicação web, em que os usuários podem criar uma sala virtual, podendo convidar seus amigos para a mesma. Nesta sala, os integrantes poderão escutar a mesma música em tempo real, tendo controle compartilhado sobre sua *playlist* - adicionando, removendo e alterando entre as músicas -, além de um meio de interação entre todos os presentes na sala.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta o referencial teórico, com explicações sobre os efeitos de se ouvir música e seus aspectos sociais. A Seção 3, lista as tecnologias utilizadas, brevemente como funcionam e a importância de cada uma para o desenvolvimento do projeto. A Seção 4, apresenta o desenvolvimento da solução, descrevendo as aplicações *backend* e *frontend* e outros detalhes técnicos. Por fim, a Seção 5 conclui o artigo e aponta possibilidades de trabalhos futuros.

2. Referencial Teórico

Esta seção aborda as partes que foram fundamentais para a elaboração deste trabalho, explicando os efeitos físicos e psicológicos ao ouvir música e também o seu aspecto social. Também são abordadas as plataformas de *streaming* de música e o porque não resolvem o problema apresentado, além da tecnologia de *sockets*, essencial para o desenvolvimento do trabalho.

2.1. Os efeitos da música

Hanser (2010), revela que escutar música agradável, está associada a um aumento na liberação de serotonina, bem como redução na liberação de beta-endorfina, um peptídeo opioidérgico relacionado ao estresse. As respostas mentais à música ativam

[4] Página oficial YouTube: youtube.com
[5] Página oficial Spotify: spotify.com
[6] Página oficial SoundCloud: soundcloud.com

o sistema nervoso simpático e parassimpático, resultando em benefícios fisiológicos e psicológicos.

A música também demonstrou aumentar a liberação de dopamina, um neurotransmissor ligado à sensação de recompensa. Essas descobertas sugerem que a música pode ter efeitos neuroquímicos e fisiológicos no corpo humano. Porém, é importante ressaltar que conforme a própria autora, os estudos ainda são limitados e mais pesquisas são necessárias para entender completamente a generalização desses efeitos em diferentes pessoas e contextos (HANSER, 2010, p. 861).

Conforme Daykin et. al. (2018), a música tem sido associada à redução da ansiedade em jovens adultos, a melhora de humor, ao bem-estar mental, qualidade de vida, autoconhecimento e também a lidar com diagnósticos de problemas de saúde. A música e o canto também têm se mostrado efetivos como estratégia para aumentar o ânimo e também reduzir o risco de depressão em pessoas idosas.

Além destes, um estudo DRM (Day Reconstruction Method) demonstrado por Västfjäll, Juslin, e Hartig, analisou 2297 relatos da rotina de 207 participantes. Destes relatos, 30% incluíam música, e em 67% destes os participantes reportaram que a música influenciou a maneira que estavam se sentindo. Quase metade relatou emoções positivas, ¼ (um quarto) relatou estado de agitação e outro ¼ (um quarto) relatou emoções negativas, como tristeza ou melancolia, demonstrando assim outros efeitos psicológicos que a música pode evocar (VÄSTFJÄLL; JUSLIN; HARTIG, 2012).

2.2. O aspecto social da música

Além de todos os efeitos mencionados na seção anterior, demonstrando como a música pode promover a liberação de serotonina e diminuição do estresse, existe outro grande fator, que é o aspecto social. Pessoas escutam música em diversas situações sociais, como encontros com amigos, bares e restaurantes e também *shows*.

De acordo com o estudo de Schäfer, T. et al., (2013), em que participaram 834 pessoas, evidenciou-se que além de escutar música por questões emocionais e para regulagem do humor, ela também desempenha o papel de conectar pessoas e fortalecer o senso de pertencimento a um grupo social. Este aspecto, caracterizado pelos autores como o segundo espectro - sendo o primeiro pelas questões emocionais e pessoais -, mostra declarações dos participantes sobre vínculos sociais e afiliação, como por exemplo, a música me ajuda a mostrar que pertenço a um determinado grupo social; a música me faz sentir conectado aos meus amigos; a música me diz como outras pessoas pensam (SCHÄFER, T. et al., 2013, p. 6).

Isso demonstra os diversos aspectos pelos quais as pessoas ouvem música, sendo eles para se aproximar de seus amigos, expressar sua identidade, seus valores e também para que melhorem suas experiências sociais.

2.3. Soluções existentes

Apesar de existirem plataformas que contemplem a necessidade de escutar música de maneira individual, as soluções não atendem a possibilidade de ouvir músicas em grupos de pessoas. A Tabela 1 compara as soluções existentes em três aspectos diferentes: capacidade de usuários escutarem música sincronizadamente, controle compartilhado da *playlist* e também a interação entre os usuários.

Tabela 1: Comparação entre plataformas existentes.

Plataforma	**Usuários ouvindo música sincronizadamente?**	**Controle compartilhado da *playlist*?**	**Interação entre usuários?**
Spotify	Sim	Sim	Não
YouTube	Sim	Não	Sim
SoundCloud	Não	Não	Sim

Olhando para estas plataformas existentes, todas atendem de maneiras diferentes a necessidade de ouvir músicas em meios digitais. Entretanto, conforme Schäffer, et al. (2013), as pessoas não escutam música exclusivamente pelo seu propósito artístico, mas também por causa de seus grupos sociais, fortalecendo laços sociais e melhorando suas relações interpessoais. Levando em consideração as soluções apresentadas na Tabela 1, nenhuma delas enfatiza o aspecto social, não possuindo unanimidade neste quesito.

O Spotify é uma plataforma de *streaming* musical, que oferece aos seus usuários milhões de músicas e *podcasts*. Ele também permite a criação de *playlists* públicas e privadas. Recentemente, este serviço lançou uma funcionalidade que permite aos seus usuários *premium* criarem sessões remotas para que possam escutar a mesma música ao mesmo, entretanto, ainda não permite com que os usuários desta sessão possam interagir livremente. O YouTube*, por* sua vez, é uma rede social que tem como foco o compartilhamento de vídeos. A plataforma possui a funcionalidade de transmissão ao vivo, na qual os usuários podem assistir a um conteúdo em tempo real, podendo interagir com o *streamer* e com outros usuários através de um chat. Porém, diferentemente do Spotify os participantes não podem controlar a mídia que está sendo reproduzida, cabendo ao *streamer* selecionar o conteúdo. Além destas plataformas, existem outras aplicações que solucionam problemas parecidos. O Moosync[7] é um *player* de música capaz de reproduzir áudios locais, do YouTube, Spotify e também SoundCloud, entretanto, os usuários não podem compartilhar suas playlists ou músicas, não possuindo qualquer interação entre pessoas em diferentes dispositivos.

2.4. Uso de *websockets* em aplicações web

Sockets são um mecanismo de comunicação bidirecional em tempo real entre um cliente e um servidor. Eles permitem a troca de dados em tempo real, sem a necessidade de requisições HTTP (HyperText Transfer Protocol) (IBM, 2021). Uma conexão via *socket* comum, segue um protocolo de eventos.

A primeira etapa é um servidor, que possui um processo sempre escutando por novas conexões, para fazer isso é necessário que o servidor deixe uma porta aberta a qual o cliente poderá se conectar. A partir disso, o servidor está pronto para ficar escutando as requisições do cliente (IBM, 2021). Ao estabelecer a conexão via *socket*, cliente e servidor podem enviar e receber dados diretamente, permitindo uma comunicação instantânea e contínua com o servidor.

O uso de *sockets* é muito útil em aplicações que precisam de atualizações constantes, como por exemplo chats devido a comunicação em tempo real. Por este motivo, essa tecnologia foi essencial para a interação entre os usuários e também

[7] Página oficial Moosync: moosync.app

garantir que todos estivessem com o menor atraso possível na reprodução de suas músicas, conforme apresentado na Seção de Desenvolvimento do Sistema.

3. Metodologia

3.1. Concepção e levantamento de requisitos da aplicação

A ideia inicial para este trabalho surgiu na disciplina de Gerência de Projetos, disciplina do quinto semestre do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do IFRS Campus Farroupilha, em sua edição do semestre 2022-01. Na ocasião, o projeto proposto pela disciplina tinha a ideia de que os estudantes, organizados em equipes, pudessem gerenciar e desenvolver um projeto ao longo do semestre. Pensando nisso, foi idealizada a aplicação YASQ (acrônimo para *yet another song queue*). Este projeto precisaria delimitar escopo, partes interessadas, requisitos funcionais e não funcionais, dentre muitas outras definições importantes para a gerência de um projeto.

Na situação mencionada, o sistema idealizado possuía os requisitos delimitados na Tabela 2 (funcionais) e Tabela 3 (não-funcionais).

Tabela 2: Requisitos funcionais delimitados para a versão 1.

	Descrição	**Atingido?**
RF0	Deve montar uma playlist para ser reproduzida.	Sim
RF1	Aceitar links de mídias de outras plataformas.	Sim
RF2	Permitir gerenciamento da fila	Não
RF3	Inclusão de novas músicas	Sim
RF4	Remoção de músicas adicionadas	Não
RF5	Avançar e retroceder músicas	Não
RF6	Sincronizar *play* e *pause* entre os usuários	Sim
RF7	Avançar para a próxima música ao final da música atual	Não
RF8	Permitir que a sala seja compartilhada entre usuários através de links	Sim

Tabela 3: Requisitos não-funcionais delimitados para a versão 1.

RNF0	Interface de usuário deverá conter o mínimo de elementos visuais possíveis.
RNF1	Interface deverá ser intuitiva e autoexplicativa
RNF2	A administração da fila ficará a encargo do criador da mesma

O sistema proposto foi desenvolvido ao longo da disciplina. Porém, o mesmo não cumpriu com todos os requisitos, fazendo com que o sistema não fosse completamente funcional. Dos requisitos mencionados, foram atingidos apenas os seguintes: *RF0*, *RF1*, *RF3*, *RF6* e também *RF8*, portanto não permitindo todo o controle necessário da *playlist*. Além de não cumprir com todos os requisitos propostos, o sistema desenvolvido também acumulou diversas dívidas técnicas, tanto na aplicação *frontend* como *backend*. Isto tornou a adição das funcionalidades faltantes bem como a resolução de *bugs* encontrados muito mais difícil.

Tendo tudo isso em vista, optou-se pelo desenvolvimento de uma nova aplicação para permitir que grupos de amigos pudessem escutar música de maneira sincronizada. Esta solução teve como base os mesmos requisitos de sua primeira versão, contando ainda com adição de novas funcionalidades, conforme os novos requisitos delimitados na Tabela 4.

Tabela 4: Requisitos funcionais adicionados para a versão 2.

RF2-0	Moderação de usuários
RF2-1	Permitir remover usuários da sala
RF2-2	Chat para interação entre participantes da sala
RF2-3	Adicionar músicas através de mecanismos de buscas e não somente links
RF2-4	Usuário deve conseguir alterar o volume do áudio sendo reproduzido

Outro ponto bastante considerado para um novo desenvolvimento foi a escolha das tecnologias anteriores, visto que o *frontend* e o *backend* foram desenvolvidos na linguagem JavaScript. Ambos foram substituídos nesta segunda versão pelo seu *superset* TypeScript, o que proporcionou uma melhor experiência de desenvolvimento e também uma aplicação *type-safe*. Por fim, outra melhoria foi o uso do ORM (Object-Relational Mapping) Prisma, criando uma nova camada que facilita *queries* ao banco de dados e a maneira com que as entidades do banco são acessadas através do código.

3.2. Tecnologias utilizadas

3.2.1. TypeScript

TypeScript é uma tecnologia *open-source* desenvolvida pela Microsoft, a qual permite adicionarmos tipagens ao JavaScript, validando o código para que não ocorram exceções em *runtime*. Por ser apenas uma linguagem de desenvolvimento, todo código escrito em TypeScript é compilado para JavaScript, entretanto as tipagens permanecem, pois foram checadas e tratadas em tempo de desenvolvimento (MICROSOFT, 2023).

Uma vantagem muito grande do TypeScript é que, por ser construído em cima do JavaScript, isto permite que qualquer código escrito nesta possa ser reaproveitado. Somando isso com o fato de que nos dias de hoje JavaScript ser uma linguagem multidisciplinar, podemos utilizar estas duas em diversos ambientes, como por exemplo

aplicações web, tanto em browsers, como servidores, e também aplicações mobile.

Esta capacidade de desenvolver programas *frontend* e também *backend* foi uma das razões pela sua escolha, pois permitiu que ambas as aplicações pudessem ser criadas utilizando a mesma linguagem acelerando e melhorando a experiência de desenvolvimento.

3.2.2. Socket.IO

A biblioteca Socket.IO é uma ferramenta que tem como principal função facilitar a comunicação bidirecional entre clientes e servidores por meio de *sockets*. Ela abstrai as complexidades dos diferentes mecanismos de transporte, no caso do JavaScript o padrão WebSockets, oferecendo uma API de fácil utilização a qual é compatível com TypeScript (SOCKET.IO, 2023).

Socket.IO apresenta recursos avançados, como recuperação automática de conexão, salas de *socket*, *callbacks* em confirmações de recebimentos, *namespaces* e broadcast de mensagens para múltiplos clientes. Estas funcionalidades avançadas foram fatores determinantes para sua escolha no projeto, visto que a comunicação via *sockets* é peça fundamental da aplicação e possuir recursos como salas de comunicação e reconexão automática, pouparam muito tempo de desenvolvimento.

Além disso, a biblioteca possui duas versões, uma para o cliente (utilizada no *frontend*) e outra para o servidor (utilizada no *backend*). Somando seu uso com TypeScript, é possível criar tipos para eventos emitidos (tanto de servidor para cliente como cliente para servidor), seus parâmetros e também retornos, fazendo com que a experiência de desenvolvimento seja aprimorada.

Esta biblioteca foi de extrema importância pois, além de ser utilizada para a troca de mensagens entre usuários de uma mesma sala, a comunicação via *socket* também foi essencial para que, ao haver qualquer mudança na playlist de músicas de uma sala, o servidor pudesse notificar todos os usuários desta sala o mais rápido possível.

3.2.3. Tecnologias *backend*

3.2.3.1. NodeJS

NodeJS, é um ambiente de execução de JavaScript, permitindo que a linguagem seja utilizada para desenvolvimento *backend*, ou seja, fora de um browser onde é comumente utilizada (OPENJS FOUNDATION, 2023a). Além disso, quando combinada com outros *frameworks*, possibilita e agiliza o desenvolvimento de API (Application Programming Interface) REST (Representational State Transfer). Esta tecnologia foi escolhida para ser utilizada no *backend* da aplicação, permitindo que um servidor esteja sempre rodando a aplicação, para que esteja pronta para receber e responder requisições HTTP e também todas chamadas via *websocket*.

3.2.3.2. Express

O Express é um *framework* minimalista e flexível para desenvolvimento de aplicações web usando o ambiente NodeJS. Ele oferece um conjunto abrangente de recursos, suportando requisições HTTP, *middlewares*, facilitando criação de rotas, criação de templates e muito mais, tudo isso possibilitando criar rapidamente APIs robustas e completas (OPENJS FOUNDATION, 2023b).

Visando estes benefícios, o *framework* foi escolhido para o desenvolvimento do

backend, pois permitiu criar o servidor responsável pelas requisições HTTPs, a camada de *middleware* responsável por tratar erros lançados pela aplicação (e também capturar exceções não tratadas) e também servir recursos estáticos, como as imagens de perfil utilizadas pelos usuários.

3.2.3.3. YouTube Data API

Um dos requisitos não atendidos na primeira versão desta aplicação foi permitir com que usuários pudessem escolher músicas através de mecanismos de buscas por textos e palavras chaves, sendo possível adicionar músicas somente por links. Pensando em solucionar isto, esta versão do sistema utiliza as APIs públicas fornecidas pela Google para buscar por conteúdos do YouTube.

A YouTube Data API V3 permite que qualquer função normalmente executada no site do YouTube seja incorporada em outras aplicações, permitindo inserção, atualização, deleção e leitura de diversos recursos (YOUTUBE DATA API, 2023). Para utilizar estas APIs foi necessária a criação de um API *key* junto ao Google Developer Console, plataforma utilizada para gerenciamento de permissões, APIs e recursos.

Possuindo a chave é possível fazer requisições HTTP normalmente para o serviço escolhido. Outro fator importante é que a Google também disponibiliza um pacote para NodeJS no repositório npm (*node package manager*) chamado *googleapis*. Este pacote facilita muito o uso de suas APIs públicas, precisando apenas instanciar o *client*, fornecer a API *key* que foi criada e selecionar qual o *endpoint* desejado, e no caso do *endpoint* de buscas, qual o termo desejado.

3.2.4. Tecnologias *backend* - Banco de Dados

3.2.4.1. MongoDB

MongoDB é um banco de dados do tipo NoSQL, ele é orientado a documentos, utilizando sintaxe parecida com JSON (JavaScript Object Notation) para armazená-los. Ele possui capacidade de escalabilidade, flexibilidade em suas consultas e também possui indexações conforme necessário (MONGODB, 2023). Um dos provedores de estes bancos de dados é o MongoDB Atlas, que cuida das questões de infraestrutura, permitindo aos desenvolvedores concentrar-se no desenvolvimento de suas aplicações.

Para o desenvolvimento deste sistema, foi utilizado o MongoDB. A escolha pelo NoSQL foi devido a experiência de desenvolvimento juntamente com o *framework Prisma* ser mais ágil. Isso deve pois, quando o ORM é utilizado com um banco de dados relacional, é necessário criar migrações, alterando tabelas, criando colunas e todas outras situações provenientes de um ambiente SQL. Entretanto, ao utilizar NoSQL, as migrações não se tornam necessárias devido a sua natureza não possuir estruturas fortemente definidas, deixando isso a encargo das aplicações que o utilizam.

Por fim, foi utilizada a infraestrutura MongoDB Atlas, por sua facilidade de configuração, em que, para a conexão da aplicação com o banco, basta fornecer a URL (Uniform Resource Locator) do servidor com o usuário e suas credenciais. Além de que, este sistema em nuvem fornece um plano gratuito até 5 GB (GigaBytes) de armazenamento, onde há persistência dos dados nestes servidores sem qualquer custo

3.2.4.2. Prisma

O Prisma é um *framework* de Object-Relational Mapping (ORM), projetado para trabalhar com JavaScript e TypeScript, simplificando a interação com bancos de dados

relacionais e não relacionais, fornecendo uma camada de abstração que permite manipular dados de forma eficiente e *type-safe* (PRISMA, 2023). O *framework* também permite definir *data models* utilizando uma linguagem de modelagem própria, onde é possível especificar a estrutura do banco de dados, seus relacionamentos, tipos e restrições.

Ao utilizar o Prisma, é possível escrever consultas e manipular dados através de uma API, sem necessidade de escrever *queries* na linguagem nativa do *driver* do banco de dados. O *framework* se encarrega de gerar o código SQL (Structured Query Language) ou NoSQL correspondente, além de lidar com a conexão e transações com o banco. Isso permitiu dar foco nas regras de negócio da aplicação, tornando a escolha do banco de dados algo muito mais flexível.

Por todas estas razões o Prisma foi escolhido para a tarefa de consultas e persistência de dados. O suporte do *framework* para MongoDB facilitou o desenvolvimento, pois sendo um banco NoSQL, não são necessárias migrações, bastando executar um comando para que o Prisma *client* esteja atualizado com as últimas alterações descritas em seu *schema*. Somado isto ao uso de TypeScript, sempre que o *schema* é atualizado, todas estas alterações são imediatamente refletidas no código inteiro da aplicação.

3.2.5. Tecnologias *frontend*

3.2.5.1. React

O React é uma biblioteca JavaScript de código aberto amplamente utilizada para construir interfaces de usuário interativas e reativas. Desenvolvido pela Meta, o React permite a criação de componentes reutilizáveis e modulares que facilitam a construção de aplicativos web escaláveis. Ele utiliza uma abordagem baseada em virtual DOM (Document Object Model), otimizando as atualizações de elementos na página, resultando em uma experiência de usuário fluida (META OPEN SOURCE, 2023).

Devido a sua fácil reutilização de componentes, boa experiência de desenvolvimento e um vasto ecossistema de bibliotecas construídas ao seu redor, o React foi utilizado para o desenvolvimento da aplicação *frontend*. Com base na escolha desta tecnologia, outras duas bibliotecas puderam ser utilizadas, Ant Design e Tanstack Query.

3.2.5.2. Ant Design

Ant Design é uma biblioteca de componentes de interface de usuário para o aplicações web. Baseada em React, ela oferece uma ampla variedade de componentes pré-construídos, como botões, formulários, tabelas e muitos outros, que podem ser facilmente integrados aos projetos (ANT GROUP, 2023). O Ant Design possui um design moderno e responsivo, criando layouts com sistemas de linhas e colunas, o que facilita a criação de designs para telas mobile e desktop.

Esta biblioteca foi integrada a *stack* do projeto por possuir componentes visualmente atrativos, facilidade na criação de designs responsivos através do seu sistema de grid e a sua grande flexibilidade para customizações. Tudo isso permite não somente estender os componentes disponíveis, mas também criar temas e paletas de cores que facilmente se adaptam aos seus designs.

3.2.5.3. Tanstack Query

O Tanstack Query é uma biblioteca JavaScript que facilita a realização de solicitações assíncronas de dados em aplicações que utilizam React, Solid, Svelte ou Vue. Com ela, é possível realizar chamadas HTTP, buscar dados de APIs e gerenciar o estado desses dados e dessas requisições de forma altamente eficiente, permitindo utilizar *fetch* diretamente do JavaScript, ou outras bibliotecas, como por exemplo Axios (TANSTACK QUERY, 2023).

Ela é extremamente útil pois oferece recursos avançados, como o cacheamento de respostas. Isso significa que as respostas das requisições são armazenadas em cache por padrão, permitindo que dados sejam recuperados do cache quando necessário, reduzindo a quantidade de requisições e melhorando o desempenho. Esta biblioteca também possui opções de invalidação de cache, *prefetching* e *refetching*, e também há a possibilidade de que as requisições sejam refeitas com base em eventos do browser *ou* intervalos pré determinados de tempo.

4. Desenvolvimento do Sistema

Esta seção apresenta os principais aspectos técnicos do desenvolvimento da solução, sendo estes: as entidades do sistema, como as músicas são reproduzidas para os usuários, o desenvolvimento das aplicações *backend* e *frontend* e por fim, como funciona a sincronia entre usuários em uma mesma sala.

A Figura 1 apresenta como as tecnologias - apresentadas na seção de Metodologia - foram utilizadas entre as aplicações desenvolvidas. O *backend* foi escrito em TypeScript, a aplicação utiliza Express para criar um servidor HTTP, a biblioteca *googleapis* para chamadas a API do Google, Socket.IO para se comunicar via *socket* com o *frontend* e Prisma como ORM para executar *queries* no banco de dados MongoDB. O *frontend* por sua vez, também foi escrito em TypeScript, utiliza React e AntDesign para construir a interface de usuário. TanstackQuery e Axios fazem as requisições HTTP e a biblioteca Socket.IO - versão *client* - se comunica através de *sockets* com o servidor.

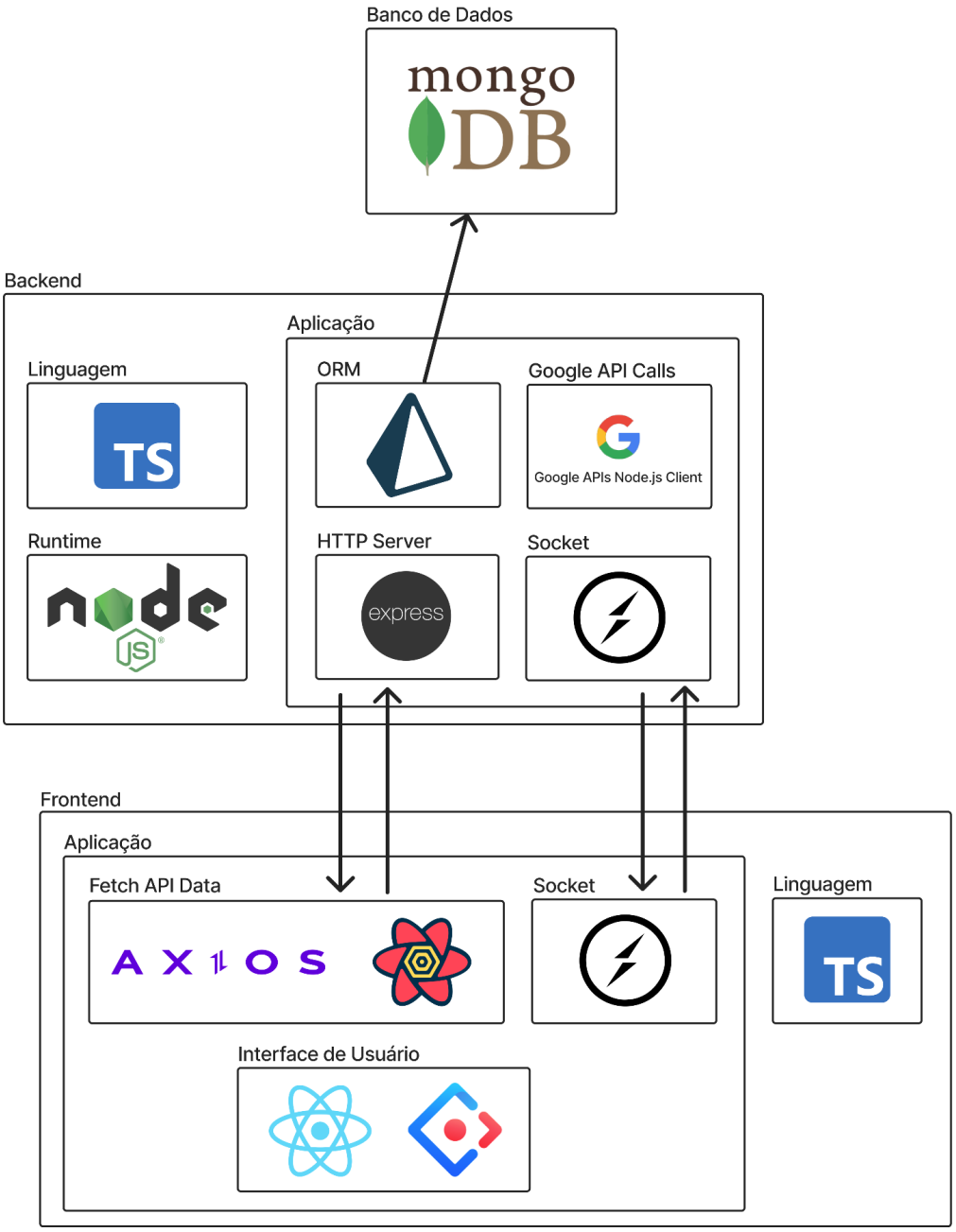

Figura 1: Diagrama exemplificando o uso das tecnologias
Fonte: Autoria própria com uso do software Figma

4.1. Entidades do sistema

Levando em conta os requisitos funcionais e não funcionais do sistema, conforme apresentados nas Tabelas 2, 3 e 4 (Seção 3), foram identificadas as seguintes entidades no sistema: usuário, sala, participação (de usuários em uma sala) e músicas. Pensando em como estas entidades seriam armazenadas no banco de dados, foi desenvolvido uma visualização do arquivo de *schema* do Prisma conforme a Figura 2, que é bastante similar a um diagrama ER (Entidade Relacionamento).

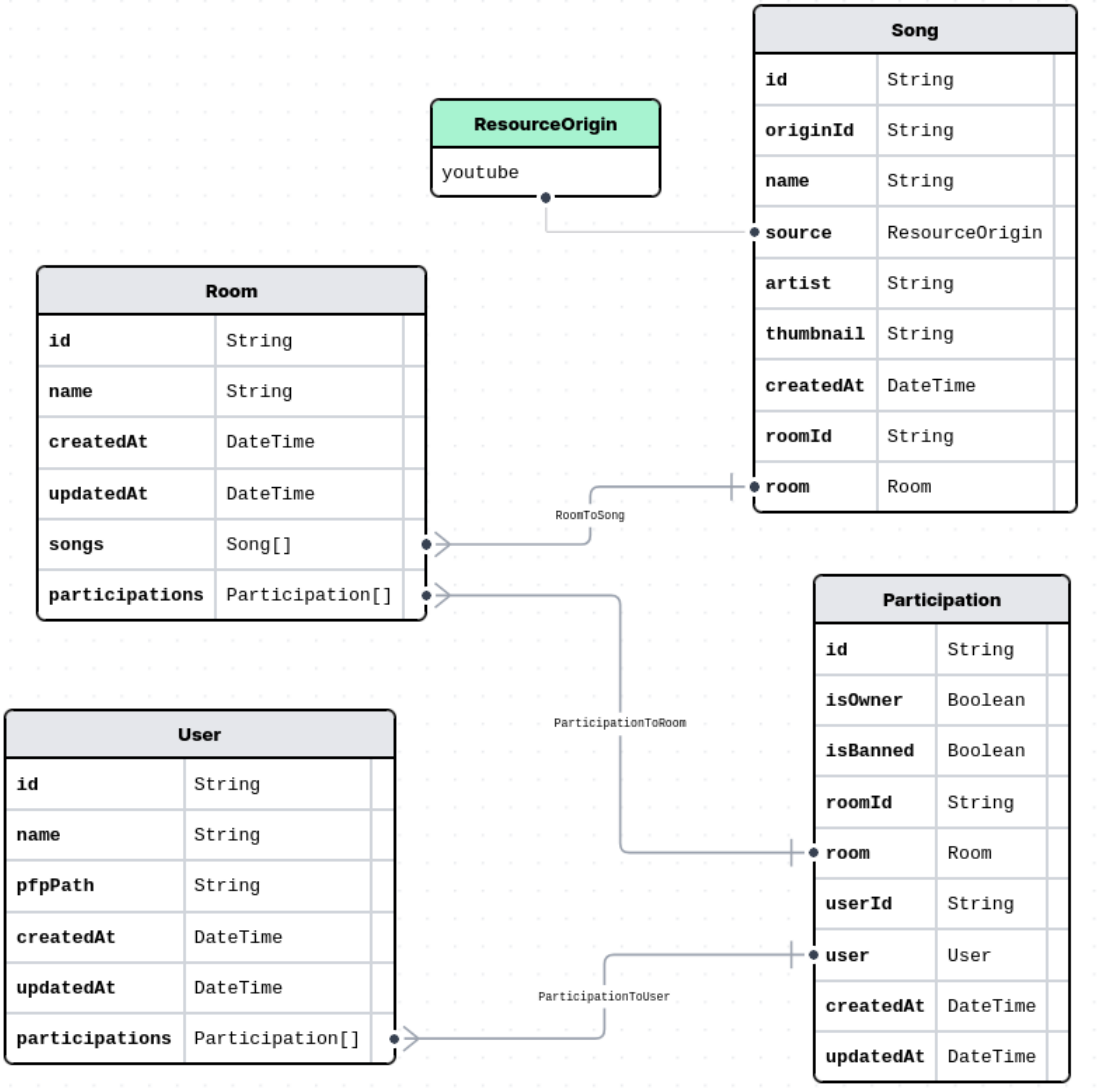

Figura 2: Diagrama do *schema* de entidades Prisma
Fonte: Autoria própria com uso do software Prismaliser

O diagrama da Figura 2 também adiciona representações para *enums* e atributos de relacionamentos criados pelo *framework*, presentes somente no nível de aplicação e não sendo armazenados no banco de dados.

Um exemplo disso são os atributos *participations* da entidade *User* e o atributo *user* da entidade *Participation*. Neste caso, a entidade *Participation* armazena no banco somente o *userId* referente ao *id* do *User*, mas possui o atributo *user* a nível de aplicação, que permite facilmente acessar o usuário relacionado aquele registro. Já o *User* não armazena nada referente a entidade *Participation* mas na aplicação, o Prisma permite acessar todas as salas que este participa através do *participations*.

A seguir são descritas, com maior nível de detalhamento, as entidades projetadas e apresentadas na Figura 2:

***Room*:** A entidade da sala virtual possui um *id* único gerado pelo banco de dados, o qual também é utilizado para gerar o link de compartilhamento. Além disso, os

outros atributos são nome, data de criação e da última atualização. Por fim, possui as relações com *Song*, que são todas as músicas presentes na *playlist* de uma sala, e com *Participation*, que são todas as participações de usuários na sala.

***Song*:** Cada música adicionada em uma *playlist* fica salva no banco de dados. Para isso, foi criada a entidade *Song*, que possui seu *id* único, mas também o *originId*, este é o *id* que identifica o vídeo, música ou mídia na sua plataforma original. Para saber qual foi a plataforma que serviu como base para esta música, é utilizado o *enum* ResourceOrigin, presente no atributo *source*. A música também tem salvo o nome de um artista, o link para a imagem da *thumbnail*, e a sua data de criação. Por fim, também possui a relação com a sala através do *roomId*.

***User*:** O usuário é uma entidade relativamente simples, possuindo seu *id* único gerado pelo banco de dados, o nome, caminho da imagem de perfil salva no servidor, e dois *timestamps* que refletem a data de criação e data da última atualização do registro. Além disso, possui o atributo *participations*, que refere a todas as salas que participa.

***Participation*:** *Participation* são todos os registros que contém as participações de usuários em salas. Um usuário pode estar em nenhuma ou muitas salas, bem como uma sala pode ter nenhum ou muitos usuários. Além de conter essas informações, a *participation* define os usuários que são moderadores através do *isOwner*, e também os usuários que foram removidos através do atributo *isBanned*. Por fim, possui os atributos de data de criação e atualização, e também as relações com *User* e *Room*.

4.2. Reprodução das músicas

O principal ponto da aplicação é a reprodução de músicas, que poderia ser implementado de diferentes maneiras. As duas implementações idealizadas foram: (1) fazer o *streaming* da faixa de áudio a partir do *backend* diretamente para cada cliente; (2) reproduzir a faixa diretamente do browser do usuário, utilizando de integrações conforme a origem da faixa selecionada.

Foi optado pela segunda alternativa, visto que facilitaria o desenvolvimento, economizaria recursos de infraestrutura e, o principal benefício, aproveitaria a eficiência de utilizar serviços e APIs especializadas neste quesito. No caso do YouTube, a ideia foi aproveitar o *iframe* disponibilizado pelo próprio YouTube para a reprodução de seus conteúdos em outras aplicações. Para o caso da plataforma Spotify, a estratégia seria utilizar a sua API em JavaScript para reproduzir o áudio. Utilizando estas alternativas, a latência é menor, pois utilizam de recursos de infraestrutura de empresas extremamente grandes e que são referência neste setor.

Durante o período de desenvolvimento da solução, foi priorizada a integração com a plataforma do YouTube, pois não requer logins e outras autenticações. Para reproduzir um vídeo do YouTube em outra página web, basta fazer o *embedding* do *iframe* no HTML da página. Esta foi a maneira utilizada para reproduzir conteúdos de origem do YouTube na aplicação, embutindo o *iframe* e não deixando-o visível para o usuário, mas ainda permitindo ao usuário dar *play*, *pause* e alterar o volume através de ações na aplicação a qual refletem as atualizações neste *iframe*.

Apesar de não terem sido implementadas na solução desenvolvida, foi identificada a viabilidade de adicionar reprodução de músicas dos serviços de *streaming* do Spotify e também do SoundCloud. O Spotify possui um SDK (Software Development Kit) chamado Web Playback, o qual permite que músicas deste serviço sejam reproduzidas em outras aplicações, contanto que para uso não comercial

(SPOTIFY FOR DEVELOPERS, 2023). O SoundCloud, por sua vez, possui APIs para buscar informações sobre faixas, artistas e *playlists*, e também possui um *widget* que pode ser adicionado através de HTML, permitindo a reprodução de suas músicas em páginas web (SOUNDCLOUD, 2023). É importante notar que para ambas as soluções - Spotify e SoundCloud -, é necessário que os usuários estejam autenticados com suas contas da plataforma que desejam utilizar.

4.3. Desenvolvimento *backend*

Para o desenvolvimento da aplicação *backend* foi criado um servidor HTTP para atender as requisições vindas do cliente (aplicação *frontend* em React). A API REST desenvolvida é responsável por fazer as operações envolvendo escritas e leituras no banco de dados. Também existe a camada de *websockets*, responsável por notificar usuários sobre alterações em sua sala e também para o envio e recebimento de mensagens entre os participantes.

A API desenvolvida foi dividida em algumas camadas, sendo as principais: *controllers, routers* e *services*. A primeira são os *routers*, cuja função é mapear as rotas chamadas através das URLs para a sua devida *controller*. A camada de *controllers* é responsável por receber as requisições HTTP vindas do cliente, verificar se todos os dados necessários para a requisição estão preenchidos, comunicar-se com a camada de serviços, e por fim responder a requisição do cliente. Por fim, a camada de *services* tem como função comunicar-se com as entidades, validar regras de negócio e utilizar da API do Prisma para ler e escrever no banco de dados. Além dessas, existe um *middleware* responsável por capturar todas as exceções lançadas pela aplicação, sendo elas intencionais ou não.

Um dos *endpoints* criados para a API foi o de buscas no serviço do YouTube, permitindo aos usuários buscarem através de textos a música desejada. Esta API utiliza-se do módulo *googleapis* para fazer *queries* no serviço YouTube Data API V3 da Google, no qual tendo uma API *key* torna fácil fazer as buscas, bastando enviar o termo de busca solicitado. Sendo assim, a *controller* recebe a requisição do usuário contendo o texto que ele deseja buscar, a aplicação utiliza sua API *key*, informa o termo de busca do usuário, e também faz alguns outros filtros. O principal destes filtros é o atributo *syndicated*, que busca somente por vídeos do YouTube que podem ser reproduzidos em *iframes* fora da plataforma original.

Além do *endpoint* de pesquisa, que foi um diferencial para a primeira versão, foram criados muitos outros, que servem para operações mais básicas da aplicação. Estes foram divididos conforme as entidades apresentadas da solução, sendo que, para cada entidade foram necessários *endpoints* para criar, ler, editar e deletar um registro no banco de dados. Todas estas operações, precisam passar pela camada de *services* que checa as regras de negócio e salva as alterações no banco.

O servidor também é responsável por armazenar em memória as informações atuais da sala, por que elas são necessárias para controlar o estado atual da sala. Sendo assim, quando um usuário executa uma ação de *play* ou *pause* na música, bem como ações de avançar ou retroceder, sendo estas ações que são utilizadas através de conexão via *socket*, não são necessárias consultas ao banco de dados. Outro detalhe é que para garantir uma única instância de cada sala em memória, foi utilizado o *design pattern* Registry (FOWLER, et al, 2002).

4.4. Desenvolvimento *frontend*

A aplicação *frontend* deste projeto tem como principais objetivos: reproduzir a música atual da fila, mostrar ao usuário informações sobre a sala em que ele se encontra e também conectá-lo a outros usuários, seja por meio de atualizações das mudanças da fila de música bem como a troca de mensagens pelo chat. Por se tratar de uma aplicação web que roda no browser de seus usuários, a base é HTML, CSS e JavaScript, construída utilizando TypeScript, React, Socket.IO, Axios, a biblioteca de componentes Ant Design e também o *framework* Tanstack Query.

O desenvolvimento dessa etapa foi dividido em componentes, interfaces e *players*. Os componentes são pequenos trechos de códigos reutilizáveis, os *players*, que são responsáveis pela reprodução das músicas, e por fim, as interfaces são sempre uma junção de componentes e representam uma página que o usuário pode acessar.

Foram desenvolvidos diversos componentes, os quais são responsáveis por mostrar o que está acontecendo na sala para os usuários. Alguns exemplos são: a barra de pesquisa, a lista de músicas, a lista de usuários, o chat, além de diversos outros. Estes são mais simples, e por vezes chamados de *dumb components*, pois não têm - ou contém minimamente - regras de negócio.

Além destes, o *player* criado também é um componente, porém possui regras de negócio e conexão via *socket* com o servidor. Este é responsável por reproduzir a música no browser do usuário, conforme mostra a Figura 3.

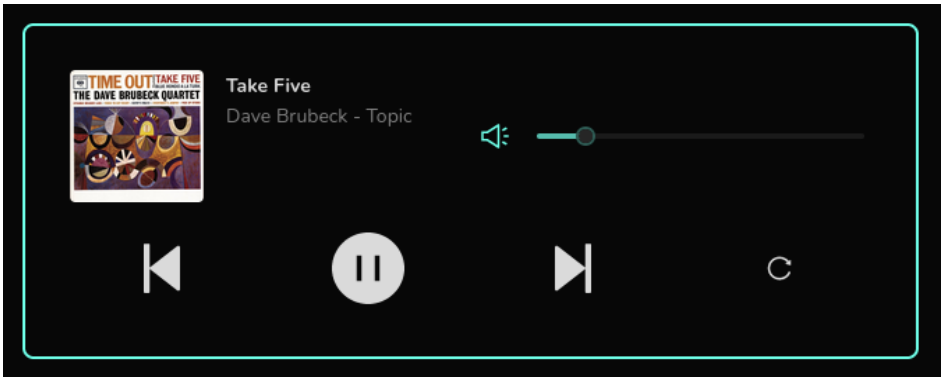

Figura 3: Componente *player* no *layout* mobile
Fonte: Autoria própria - *screenshot* da aplicação desenvolvida

Tendo em consideração que a música pode vir de diversas fontes - apesar de atualmente somente o YouTube ser suportado -, a experiência para o usuário deve ser sempre a mesma. Pensando nisso, foi adicionado o *iframe* do YouTube, mas todos os controles desse elemento são abstraídos no componente de *player* customizado, que além de ser o mesmo independente da origem da música, também contempla as diversas validações da aplicação.

Também foram criadas duas interfaces principais, as quais são responsáveis por utilizar os componentes, fazer as requisições para a API e validar regras de negócio, sendo elas: (1) a interface que permite ao usuário criar ou entrar em uma sala, e (2) a interface da sala, que possui o chat, *player* de música, dentre outras funcionalidades.

Conforme a Figura 4, a página inicial é bastante simples, possuindo apenas uma breve descrição da aplicação e possuindo um campo de texto para que o usuário possa dar um nome a sua sala ao criá-la, ou então utilizar o código de uma outra sala que

esteja criada.

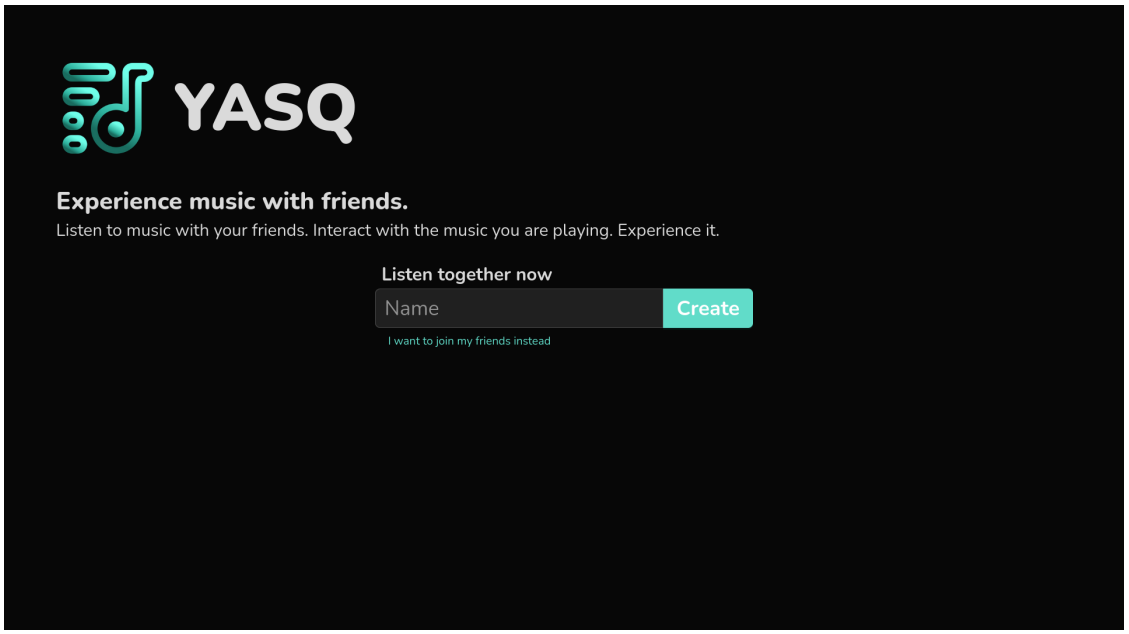

Figura 4: Interface inicial para criar ou entrar em uma sala virtual
Fonte: Autoria própria - *screenshot* da aplicação desenvolvida

Quando o usuário entra em uma sala após esta ter sido recém criada, ela ainda estará vazia, sem nenhuma música na fila ou outros participantes (Figura 5). Após entrar na sala, ele poderá utilizar a barra de pesquisa no topo para pesquisar pela música que deseja ouvir, ou inserir um link neste mesmo campo para adicionar diretamente a faixa que deseja. No lado superior direito existem dois botões e um ícone com a foto de perfil do usuário. O botão do lado esquerdo copia o link da sala para a área de transferência do usuário, permitindo que ele compartilhe a mesma. O outro botão serve para o usuário sair da sala, o qual abrirá um *pop up* solicitando confirmação desta ação.

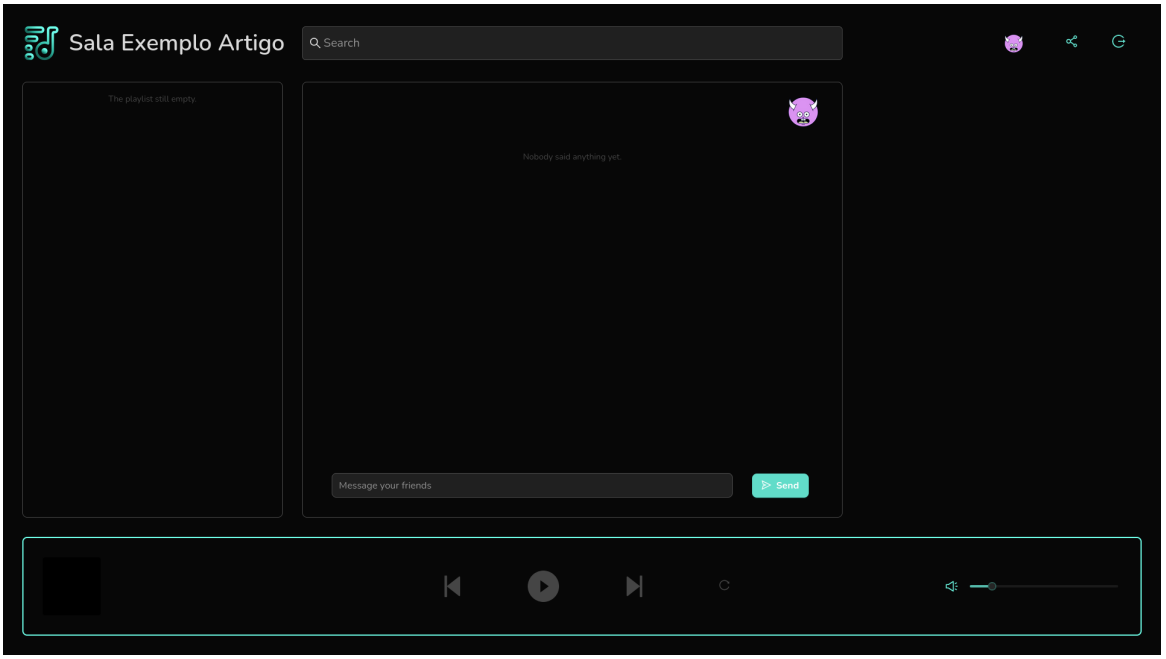

Figura 5: Interface da sala ao recém criá-la
Fonte: Autoria própria - *screenshot* da aplicação desenvolvida

Após outros usuários entrarem na sala, e também adicionar músicas na fila, a sala ficará com a aparência parecida a Figura 6. No lado esquerdo da tela o usuário possui uma lista com todas as músicas que estão na fila. A música que está sendo reproduzida atualmente possui uma borda da cor primária da aplicação. Caso passe com o cursor por cima de um item da lista, aparecerá um botão com ícone de lixeira, utilizado para remover músicas da fila.

No meio da tela o usuário tem o chat da aplicação, que além de servir para mandar mensagens para todos os usuários presentes na sala, também notifica das ações da sala, como adição e remoção de músicas, se alguém passou para a música anterior ou para a próxima, ou se alguém deu *play* ou *pause* na música atual. Caso clique na lista de fotos dos usuários que fica acima do chat, ele terá uma lista completa de todos os participantes, e caso seja um moderador, poderá através desta remover outros usuários que não sejam moderadores.

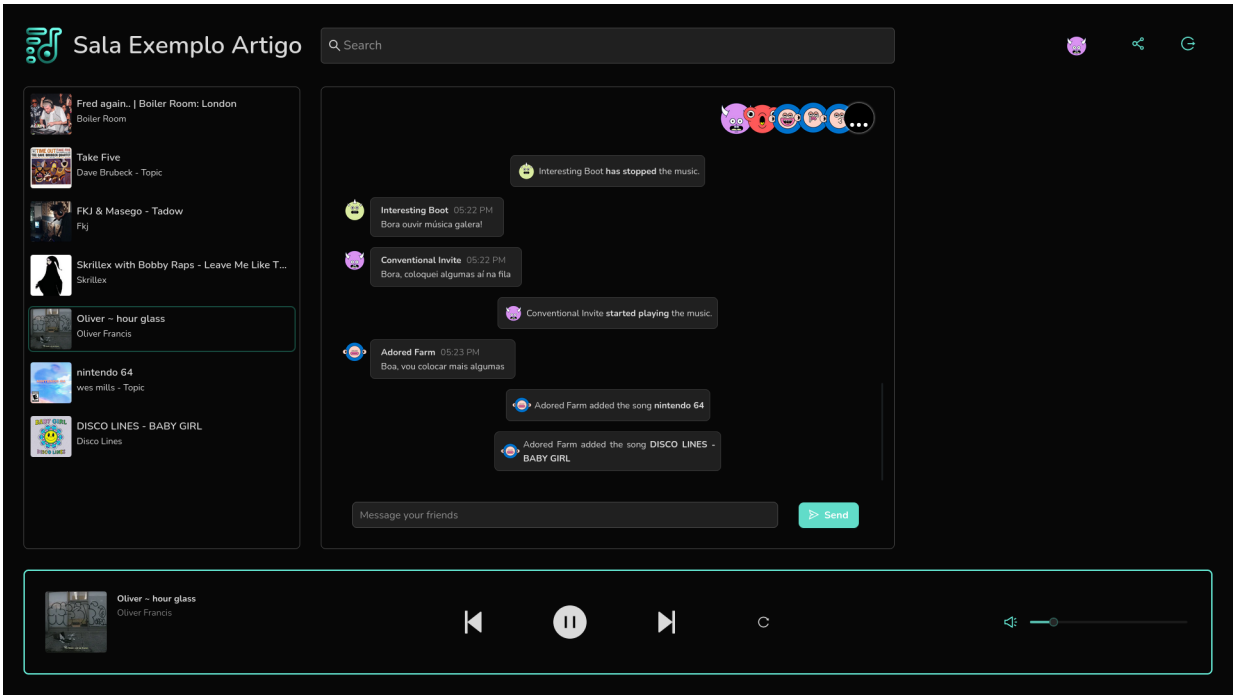

Figura 6: Interface da sala com usuários e músicas
Fonte: Autoria própria - *screenshot* da aplicação desenvolvida

4.5. *Websockets* para sincronia entre usuários

Para facilitar a comunicação via *websockets* foi utilizada a biblioteca Socket.IO, que foi de extrema importância para garantir que os usuários estivessem sempre o mais sincronizados possível.

A comunicação via *socket* é estabelecida assim que o usuário entra em uma sala. Após isso ele recebe um novo evento do servidor, informando se há alguma música no *player*, se está parada ou tocando, e qual o tempo transcorrido na música atual. Com essas informações o *player* deste usuário estará sincronizado com as mesmas informações que os outros participantes.

Ao pressionar os botões de *play* ou *pause* em uma música, foi aplicada uma regra para garantir que todos os participantes da sala estivessem no tempo mais próximo possível. Quando um usuário efetua esta ação no *frontend*, a única coisa que ocorre é a emissão de um evento via *socket* para o servidor - representado pela linha azul na Figura 7 -, indicando a ação desejada, ou seja, efetivamente o estado da sala ainda não mudou para este ou qualquer outro usuário. Quando o servidor recebe este evento, ele replica o evento via *socket* para todos os participantes da sala, informando que o estado atual da sala foi alterado - conforme as linhas verdes na Figura 7 -, e ao receber este evento do servidor, é que a música é então pausada ou volta a tocar. Esta técnica também foi aplicada para os eventos de avançar e retroceder na música atual da *playlist*, buscando diminuir ao máximo a diferença de tempo entre os participantes.

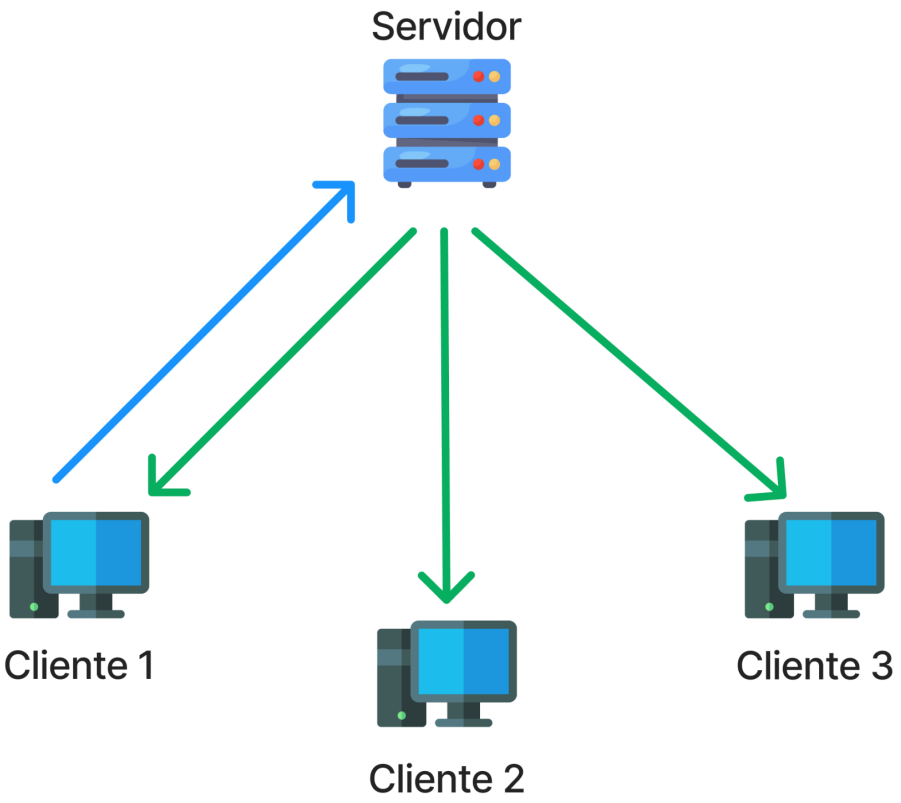

Figura 7: Representação dos eventos via *socket* para alterações no estado da sala
Fonte: Autoria própria

Quando outras ações ocorrem na sala, como por exemplo, adição ou remoção de uma música ou quando um usuário entra ou sai da sala, o *socket* também é utilizado. No momento em que estas ações ocorrem via requisição HTTP, a aplicação *backend* também emite um evento para todos os participantes da sala, indicando a ação ocorrida. Por exemplo, se alguém adiciona uma música, na chamada para a API, todos usuários recebem um evento via *socket* que a *playlist* foi alterada. Ao receber este evento, a aplicação *frontend* invalida o cache - que é controlado pela biblioteca Tanstack Query -, acionando uma nova requisição HTTP para o servidor para buscar pela *playlist* atualizada.

5. Conclusões

Este artigo apresentou diversas soluções no que se propõe a escutar música de maneira individual. Também foi demonstrado que, no contexto da música, pessoas não escutam-a somente por entretenimento mas também pelo seu valor social e pelas conexões com suas amizades.

Pensando nisso, foi elencado o problema de que não existiam soluções que permitissem grupos de amigos escutarem música de maneira sincronizada, conjunta, e promovendo a interação entre todos participantes. Por isso, foi proposta e desenvolvida uma aplicação web, *frontend* e *backend*, que resolve este problema através de uma sala virtual, que além de reproduzir faixas de áudio de maneira sincronizada também permite que seus usuários possam interagir através de um chat.

A solução apresentada sana esta dor, porém, ainda conta com espaço para melhorias, dando abertura para adição de novas funcionalidades em futuros desenvolvimentos, como por exemplo, adicionar suporte a reprodução de música de outras plataformas além do YouTube - como Spotify e SoundCloud -, autenticação de usuários para consequentemente contemplar as outras plataformas, criar uma visualização de todas as salas que o usuário participa, permitindo-o escolher a sala ao entrar na aplicação e também armazenamento das mensagens enviadas em um banco de dados, possibilitando usuários navegarem pelo histórico de mensagens.

Referências

ANT GROUP. **The world's second most popular React UI framework.** Disponível em: <ant.design>. Acesso em: 25 de abril de 2023.

DAYKIN, N. et al. What works for wellbeing? A systematic review of wellbeing outcomes for music and singing in adults. **Perspectives in public health**, v. 138, n. 1, p. 39–46, 2018.

FOWLER, M. et al. Registry. **Patterns of Enterprise Application Architecture.** v. 1, p. 480, 2002.

HANSER, S. B. Music, health, and well-being. Em: **Handbook of Music and Emotion: Theory, Research, Applications**. London, England: Oxford University Press, 2010. p. 849–877.

IBM. **How sockets work, IBM Documentation**, 14 de abril de 2021a. Disponível em: <www.ibm.com/docs/en/i/7.2?topic=programming-how-sockets-work>. Acesso em: 04 de junho de 2023.

IBM. **Understanding sockets concepts, IBM Documentation**, 22 de março de 2021b. Disponível em: <www.ibm.com/docs/en/zos/2.2.0?topic=concepts-understanding-sockets>. Acesso em: 15 de julho de 2023.

META OPEN SOURCE. **React: The library for web and native user interfaces.** Disponível em: <react.dev>. Acesso em: 09 de junho de 2023.

MICROSOFT. **TypeScript is JavaScript with syntax for types.** Disponível em: <www.typescriptlang.org>. Acesso em: 08 de junho de 2023.

MONGODB. **MongoDB: The developer data platform.** Disponível em: <mongodb.com>. Acesso em: 08 de junho de 2023.

OPENJS FOUNDATION. **Express: Fast, unopinionated, minimalist web framework for Node.js.** Disponível em: <expressjs.com>. Acesso em: 10 de junho de 2023a.

OPENJS FOUNDATION. **Node.js is an open-source, cross-platform JavaScript**

runtime environment. Disponível em: <nodejs.org>. Acesso em: 08 de junho de 2023b.

PRISMA. **Next-generation Node.js and TypeScript ORM.** Disponível em: <prisma.io>. Acesso em: 10 de junho de 2023.

SCHÄFER, T. et al. The psychological functions of music listening. **Frontiers in psychology**, v. 4, p. 511, 2013.
Disponível em: <www.frontiersin.org/articles/10.3389/fpsyg.2013.00511/full>

SOCKET.IO. **Bidirectional and low-latency communication for every platform.** Disponível em: <socket.io>. Acesso em: 10 de junho de 2023.

SOUNDCLOUD. **API Guide - SoundCloud Developers.** Disponível em: <developers.soundcloud.com/docs/api/guide#playing>. Acesso em: 12 junho de 2023.

SPOTIFY FOR DEVELOPERS. **Web Playback SDK: Create a new player and stream Spotify content inside your website application..** Disponível em: <developer.spotify.com/documentation/web-playback-sdk>. Acesso em : 05 de maio de 2023.

TANSTACK QUERY. **Powerful asynchronous state management for TS/JS, React, Solid, Vue and Svelte.** Disponível em: <tanstack.com/query/v3/>. Acesso em: 10 de junho de 2023.

VÄSTFJÄLL, D.; JUSLIN, P. N.; HARTIG, T. Music, subjective well being, and health: The role of everyday emotions. **Music, Health, and Wellbeing**. [s.l.] Oxford University Press, 2012. p. 405–423.

YOUTUBE DATA API. **Add YouTube functionality to your app.** Disponível em: <developers.google.com/youtube/v3/>. Acesso em: 27 de maio de 2023.