Streamaster

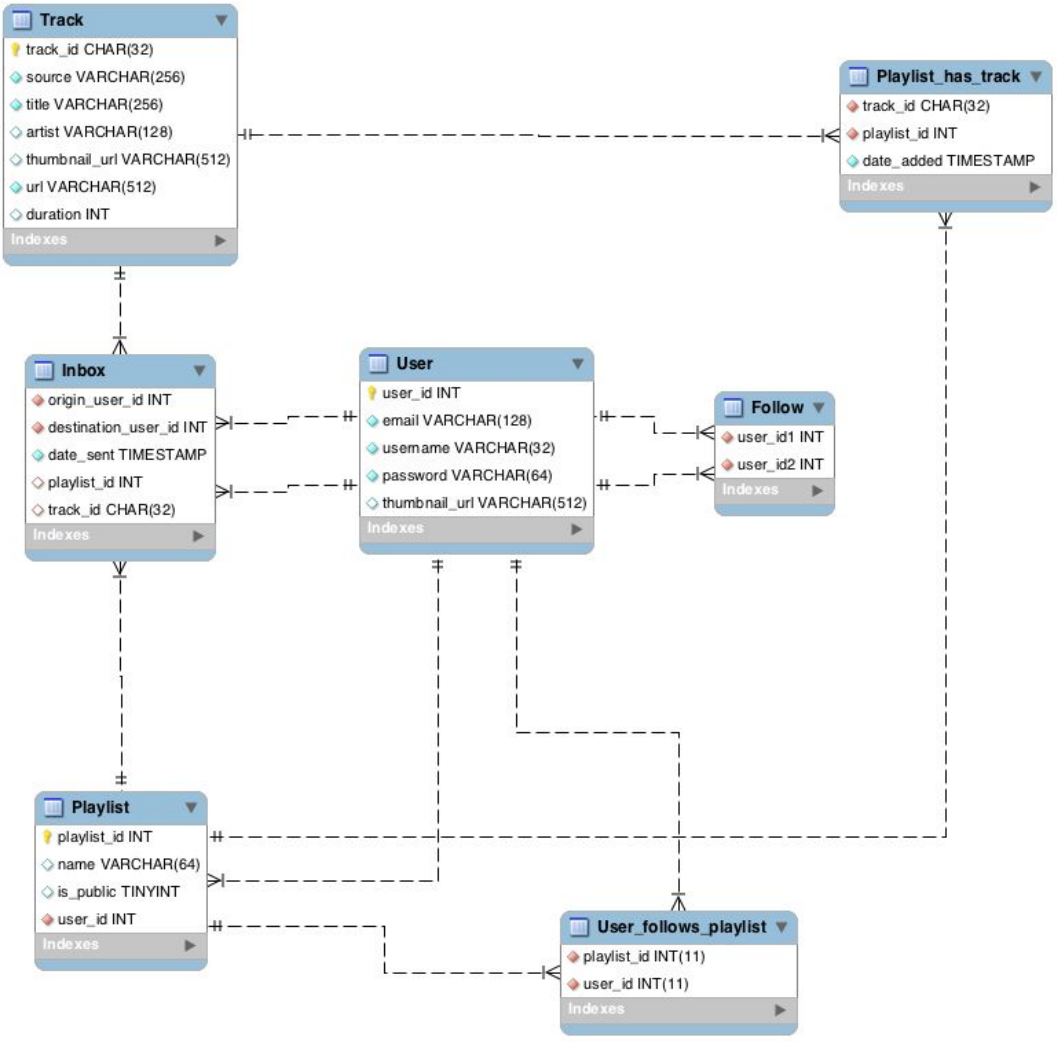
O que é o Streamaster?

O Streamaster é uma plataforma de streaming de música que agrega os resultados de uma busca dentre 3 serviços: Spotify, Soundcloud e Youtube. A plataforma foi projetada utilizando ReactJS e inclui um player que possibilita o usuário de reproduzir as músicas diretamente do app.

Nesse projeto, adicionamos novas funcionalidades ao Streamaster através da criação de uma API própria. Através desta, é possível que o usuário cadastre-se no App, utilizando um email, um username e uma senha. Já cadastrado, é possível, além de playlists, seguir outros usuários, seguir playlists e compartilhar músicas com quem o usuário segue.

Modelagem Entidade - Relacionamento

Para a base de dados, a seguinte modelagem de Entidade - Relacionamento foi utilizada:



Dicionário da base:

User - Guarda os usuários e suas informações

user_id - ID exclusivo de cada usuário
email - Email exclusivo de cada usuário
username - Username identificador exclusivo
password - Senha
thumbnail_url - URL opcional de uma thumbnail

Follow - Guarda relações de seguidor/seguidores

user_id1 - ID do usuário 1
user_id2 - ID do usuário 2

Playlist - Guarda as playlists existentes no serviço

playlist_id - ID única de cada playlist
name - Nome da playlist
is_public - Flag que indica se a playlist é publica ou privada
user_id - ID do usuário dono da playlist

Track - Guarda as informações das músicas ou videos

track_id - ID único de cada track, criado com hashing
source - String que indica a fonte eg. “spotify" | “soundcloud"
title - Título da música
artist - Artista da música
thumbnail_url - Url opcional de thumbnail para a track
url - Url que será usada para o streaming
duration_ms - Duração, em ms, da faixa

Playlist_has_track - Relaciona as playlists com as tracks

id_track - Chave para a track na tabela Track
id_playlist - Chave para a playlist que a Track pertence
date_added - Timestamp automática da data de adição

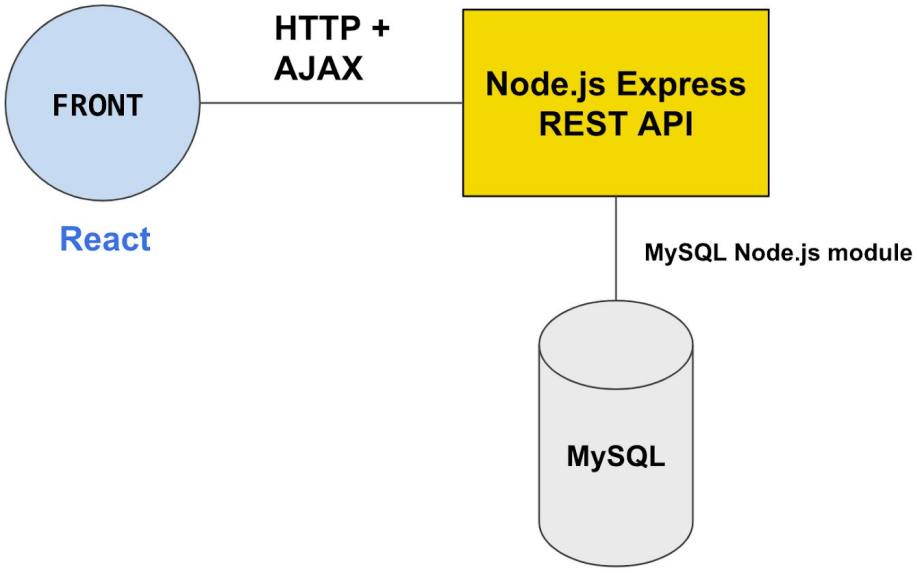
Inbox - **Caixa de entrada para músicas compartilhadas**

origin_user_id - Chave do usuário rementente da mensagem
destination_user_id - Chave do usuário destinatário
date_sent - Timestamp automático da data de envio
playlist_id - ID da playlist que está sendo compartilhada
track_id - ID da track que está sendo compartilhada

User_follows_playlist - Relaciona uma playlist com usuários que a seguem

playlist_id - Chave identificadora da playlist
user_id - Chave identificadora do usuário que segue a playlist

Infraestutura do projeto



- **Documentação de uso do sistema (Ambiente de produção)**

  Antes de qualquer coisa é necessário rodar os scripts de criação do banco de dados

  Para utilizar o ambiente de produção, basta buildar o front end

  ```
  $ cd streamaster/front
  $ npm run build
  ```

  Após executar tais comandos serão gerados os arquivos estáticos no diretório `streamaster/front/build`, e o express já está preparado para serví-los na porta 5000

  ```
  $ cd streamaster
  $ nodemon server.js OU node server.js
  ```

  Acessar http://localhost:5000 e consumir a aplicação.

Caso o servidor não seja iniciado corretamente verificar as credenciais de conexão com o MySQL no começo do arquivo server.js.

- **Documentação de uso do sistema (Ambiente de desenvolvimento)**

  Para utilizar o sistema no ambiente de desenvolvimento, é necessário:

  1) Rodar os scripts de criação do banco de dados
  2) Instalar as dependências do front e do back:

```
cd streamaster
npm install
cd streamaster/client
npm install
```

  Executar:

```
back  :
      cd streamaster
      nodemon server.js OU node server.js
front :
      cd streamaster/client
      npm start
```

  3) Abrir http://localhost:3000 em um Browser e usar o serviço

API REST - Endpoints e documentação

Método	Endpoint	Descrição	Entrada	Retorna

POST	/user	Registra novo usuário	body : {
			email: string
			username: string
			password: string
			file: base64
			}
				SUCESSO 200:
				user = {
				user_id: int
				username: string
				thumbnail_url: string
				}
				FALHA: 409

POST	/login	Faz login no serviço	body: {
			email: string
			password: string
			}
				SUCESSO 200:
				user = {
				user_id: int
				username: string
				thumbnail_url: string
				}
				FALHA: 404

GET	/user	Procura por usuários com substring = query	url: {
			q: string
			}
				SUCESSO 200:
				[user1, user2...]
				ou
				[ ]

POST	/playlist	Cria uma nova playlist e retorna o novo ID atribuído a esta.	body: {
			user_id: int
			}
				SUCESSO 200: {
				playlist_id: int
				}

PATCH	/playlist	Muda o nome de uma playlist	body: {
			playlist_id: int
			new_name: string
			}
				SUCESSO 200
				FALHA 404

GET	/user/{user_id}/friend	Retorna os seguidores e os seguindo	url: {
			id: int
			}
				SUCESSO 200 {
				following: [user1...]
				followers: [user4…]
				} ou {
				following: [ ]
				followers: [ ]
				}
				FALHA 404

POST	/playlist/follow	Segue uma playlist	body: {
			user_id: int
			playlist_id: int
			}
				SUCESSO 200
				FALHA 404

DELETE	/playlist/follow	Deixa de seguir uma playlist	body: {
			user_id: int
			playlist_id: int
			}
				SUCESSO 200
				FALHA 404

GET	/user/{user_id}/playlist	Retorna todas as playlists de um usuario	url: {
			user_id: int
			}
				SUCESSO 200
				[playlist1: {
				...playlist_info
				songs: [...songs]
				}, playlist2...]
				ou
				[ ]
				FALHA 404

DELETE	/playlist/{playlist_id}	Deleta uma playlist	url: {
			id: int
			}
				SUCESSO 200
				FALHA 404

DELETE	/playlist/{playlist_id}/{track_id}	Deleta uma música de uma playlist	url {
			playlist_id: int
			track_id: int
			}
				SUCESSO 200
				FALHA 404

PUT	/playlist/{playlist_id}	Insere uma música em uma playlist	url {
			id: int
			}
			
			body {
			playlist_id: int
			song: {
			source: string
			title: string
			artist: string
			thumbnail_url: string
			url: string
			duration_ms: int
			}
			}
				SUCESSO 200 {
				track_id: int (hash)
				}
				
				FALHA 404

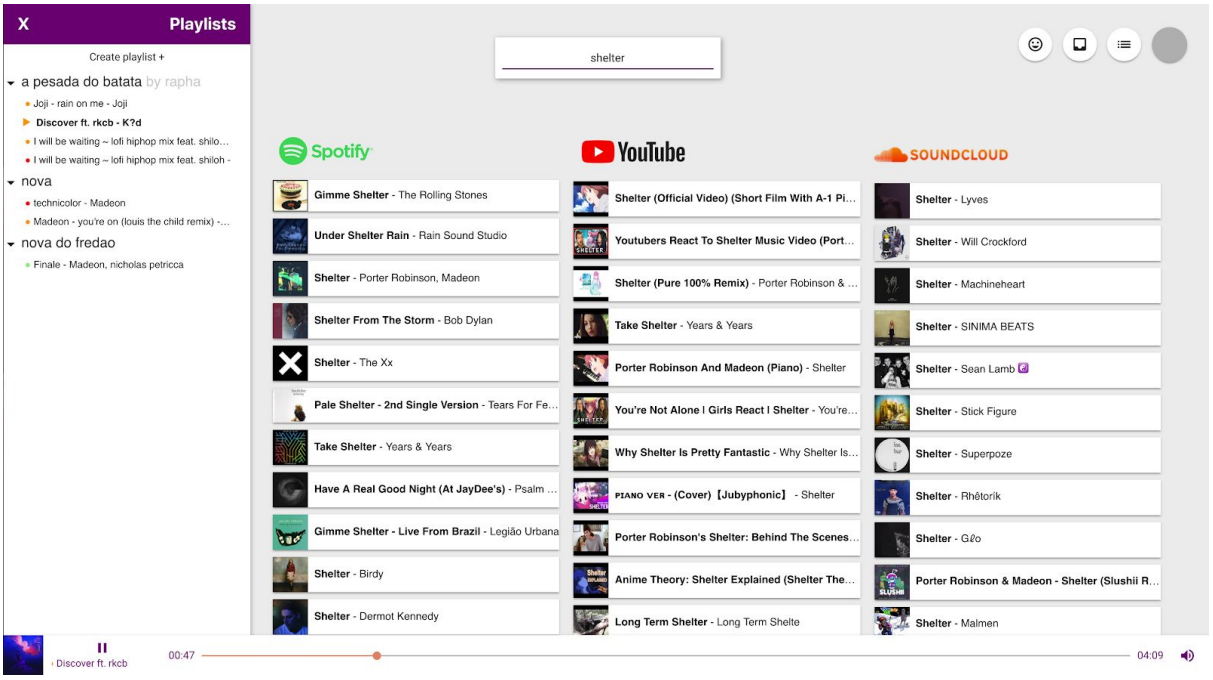
POST	/user/friend	Faz com que o user_id1 siga um usuário user_id2	body {
			user_id1: int
			user_id2: int
			}
				SUCESSO 200
				FALHA 404

DELETE	/user/friend	For com que o user_id1 deixe de seguir o user_id2	body {
			user_id1: int
			user_id2: int
			}
				SUCESSO 200
				FALHA 404

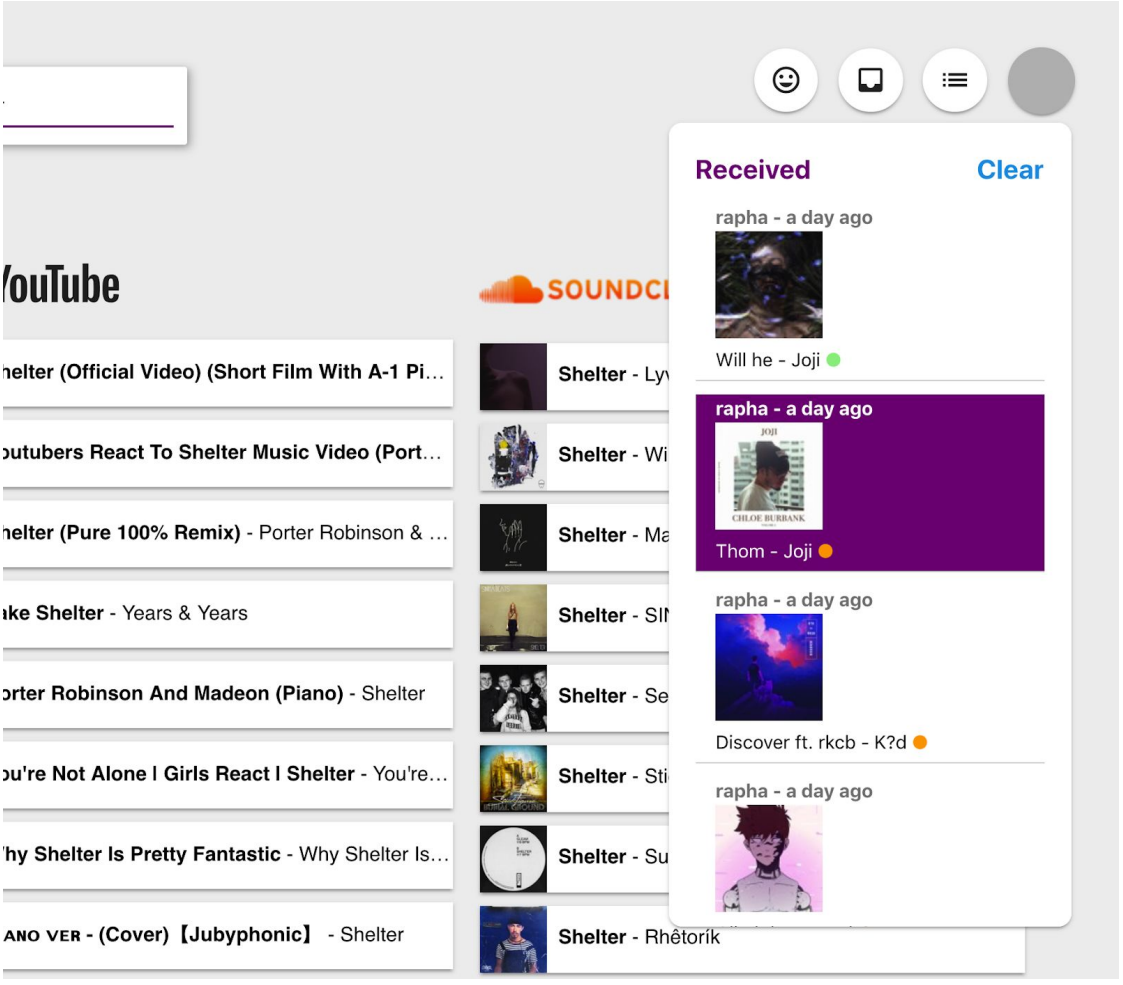
POST	/api/user/inbox	Envia uma track para o inbox de um usuário	body {
			origin_user_id: int
			destination_user_id: int
			song: song
			}
				SUCESSO 200
				FALHA 404

GET	/api/{user_id}/inbox	Pega as mensagens no Inbox de um usuário	url {
			id: int
			}
				SUCESSO 200 {
				[messages…]
				message: {
				artist string
				date_sent date
				destination_user_id int
				duration_ms int
				email string
				origin_user_id int
				playlist_id int
				source string
				thumbnail_url string
				title string
				track_id int
				url string
				user_id string
				username string
				}
				FALHA 404

DELETE	/api/{user_id}/inbox	Limpa o Inbox de um usuário	body {
			user_id: int
			}
				SUCESSO 200
				FALHA 404



Screen Shot da interface de playlists



Screen Shot da interface de Inbox