

UNIVERSIDADE FEDERAL DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO
PROGRAMAÇÃO ORIENTADA A OBJETOS I

LEONARDO DE SOUSA MARQUES
23202501, INE5402, 2023/2

RELATÓRIO DE PROJETO:
PLAYLIST GENERATOR

Relatório realizado por Leonardo de Sousa
Marques para o Trabalho Final da disciplina
INE5402, na UFSC.

FLORIANÓPOLIS, 2023.

SUMÁRIO

1. INTRODUÇÃO.....	3
2. PRINCIPAIS BIBLIOTECAS.....	3
3. CLASSE PRINCIPAL.....	4
4. FUNÇÕES E JANELAS.....	4
4.1. DEF init.....	4
4.2. Def initial_frame.....	5
4.3. Def second_frame.....	6
4.4. Def generate_playlist.....	8
4.5. Def thid_frame e open_playlist.....	10
5. EXEMPLO DE FUNCIONAMENTO.....	11
6. REFERÊNCIAS.....	13

1. INTRODUÇÃO

O seguinte relatório tem como objetivo apresentar o funcionamento do “Playlist Generator”, um programa criado para preencher o trabalho final disposto pela disciplina de Programação Orientada a Objetos I, da UFSC.

A ideia do projeto surge com o intuito de criar uma aplicação que seja útil no dia-a-dia e a fim de facilitar o trabalho humano. Neste caso, com o Playlist Generator, ao digitar o nome de artistas ou bandas e uma certa quantidade de músicas, além do nome desejado para a coletânea, é possível gerar, automaticamente, uma playlist no aplicativo *Spotify*. Desse modo, o programa tem vários fins, podendo então ser utilizado para uso pessoal e para descobrir novos artistas, quanto para criar playlists de forma fácil para uma festa, por exemplo.

Ao longo do relatório, busca-se realizar uma análise dos pontos mais importantes do código escrito na linguagem de programação Python para que possa-se explicar o aplicativo e, também, relacionar com os temas da Programação Orientada.

2. PRINCIPAIS BIBLIOTECAS

Primeiramente, é necessário introduzir as principais bibliotecas utilizadas para a produção do projeto, sendo essas a Tkinter e a Spotipy, em que ambas são utilizadas para implementação de programas na linguagem Python.

Com o auxílio da Tkinter, é possível criar GUIs (Interface Gráfica do Utilizador). Desse modo, pode-se criar diversos *widgets*, como títulos, caixas de texto e botões. Como este projeto necessita da interação com o usuário para seu funcionamento, tais objetos foram amplamente utilizados. Além disso, é possível definir o tamanho que janela de interação terá (*geometry*), o título da janela (*title*), se será possível maximizar ou ajustar o tamanho da tela ou não (*True* ou *False* para cada dimensão na função *resizable*). Para este projeto, optou-se por utilizar uma janela de dimensões 400x550 e que não fosse possível reajustar seu tamanho.

Já a biblioteca Spotipy foi utilizada para facilitar o processo de pesquisa e autenticação dentro do aplicativo *Spotify*. Como será demonstrado ao longo deste relatório, foram utilizadas funções como “*sp.search*”, “*sp.user_playlist_create*”, “*sp.artist_albums*” e outras para a realização do programa.

Por fim, claramente, é necessário ter instalada a linguagem Python para que se possa utilizá-las. Ademais, também foram utilizadas as bibliotecas Random e Webbrowser, que geralmente já vêm instaladas juntamente à linguagem.

Para fazer a instalação do Python, pode-se baixar diretamente do site oficial: *python.org.br*. Já para o Tkinter e Spotipy, utiliza-se diretamente o prompt de comando — ou o terminal do editor de texto de escolha.

Tabela 1 - Instalação das bibliotecas Tkinter e Spotipy

Tkinter	pip install tkinter (Windows) pip3 install tkinter (Mac)
Spotipy	pip install spotipy (Windows) pip3 install spotipy (Mac)

3. CLASSE PRINCIPAL

O conceito de classes dentro da Programação Orientada a Objetos é dado pela relação entre objetos com características em comum. No caso desta aplicação, utilizou-se apenas uma classe principal chamada “Aplicativo”, de modo que, através dela, o programa com a interface gráfica é rodado. Dentro desta classe, o código é dividido em diversas funções (*def*) para que as instruções possam ser realizadas. Desse modo, é necessário utilizar a indicação *self* para que as variáveis do código possam interagir entre si, quando necessário, mesmo estando em diferentes funções do código.

4. FUNÇÕES E JANELAS

Dentro da classe principal Aplicativo, o código é separado em seis funções *def*, sendo elas: *init*, *initial_frame*, *second_frame*, *generate_playlist*, *third_frame* e *open_playlist*.

4.1. DEF init

O principal objetivo desta função é definir os parâmetros iniciais da aplicação e demais variáveis. Foram atribuídos os tamanhos da janela, título, variáveis que serão utilizadas na geração da playlist e a definição de que a *initial_frame* será a página 1 do programa. Observa-se no fragmento abaixo esta parte do código, juntamente à classe.

Fragmento 1 - Classe e função inicial

```
#Classe do Aplicativo/Programa:
class Aplicativo:
    #Configurações de tela e variáveis self. do programa:
    def __init__(self, root):
        self.window = root
        self.window.title('Playlist Generator')
        self.window.geometry("400x550")
        self.window.resizable(False, False)
        self.artists = None
        self.num_songs = None
        self.playlist_name = None
        self.playlist_url = None
        self.initial_frame()
```

4.2. Def initial_frame

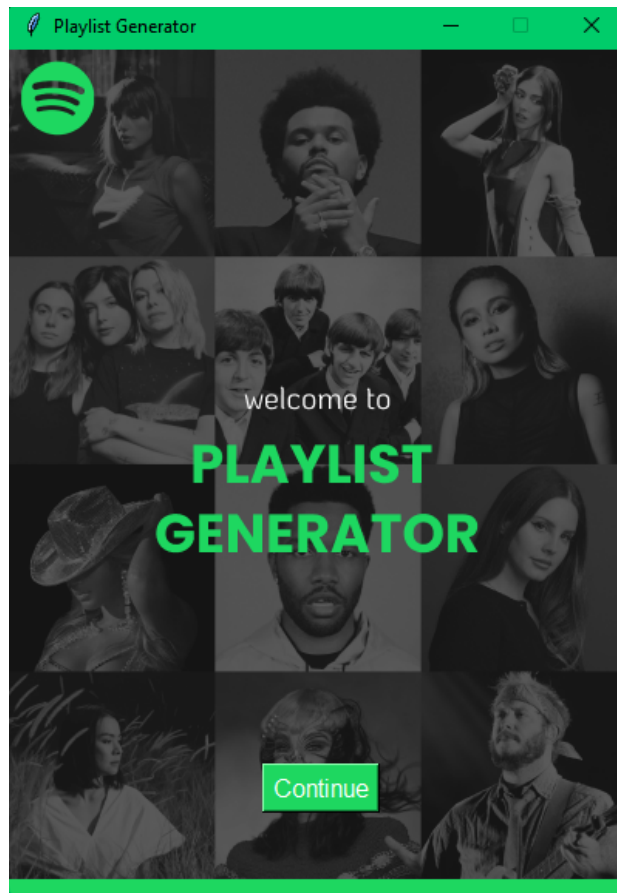
Esta é a primeira página apresentada pela janela quando o aplicativo for inicializado. Em termos de aplicação, sua única função é introduzir o programa para o usuário e, quando ele decidir usá-la, irá pressionar o botão (função *button* do tkinter) e ir para a segunda página. No fragmento de código abaixo, observa-se como é feita essa lógica.

Fragmento 2 - Código para a página inicial

```
#Página Inicial
def initial_frame(self):
    #Configurações de Background
    self.background = tk.PhotoImage(file="images/bg1.png")
    self.background_label = tk.Label(self.window, image=self.background)
    self.background_label.place(relwidth=1, relheight=1)
    self.initial_frame = tk.Frame(self.window)
    #Tipografia
    self.Poppins = font.Font(family="Poppins", size=12)
    #Botão de "Continue"
    continue_button = Button(text="Continue", font=self.Poppins, bg="#1ED760", fg="white",
command=self.second_frame)
    continue_button.place(relx=0.60, rely=0.90, anchor="se")
```

Tendo feito a implementação, quando o usuário rodar o programa, esta será a primeira janela com que ele terá contato e que é descrita pelo código acima:

Figura 1 - Página 1



4.3. Def second_frame

A segunda página é a GUI mais importante do código, pois é o local em que o usuário irá interagir com o programa para solicitar quais artistas estarão na playlist e quantas músicas a mesma terá. Para fazer isso, utilizou-se a lógica de que as janelas são estruturadas a partir de grids (ou matrizes), em que os objetos são posicionados em determinadas linhas e colunas. Ademais, a função “Entry” foi necessária para que as caixas de texto fossem criadas e, como essas informações serão utilizadas para a geração da playlist, nota-se que as variáveis “*artists*”, “*num_songs*” e “*playlist_name*” foram declaradas previamente em *init*. Por fim, dá-se ênfase para o botão “*Generate*”, que é utilizado para ativar a próxima função e que será utilizada para, de fato, gerar a coletânea. No código abaixo, observa-se a lógica para a disposição das caixas e algumas configurações de posicionamento utilizadas, bem como a ideia para “esconder” a página um e exibir a segunda.

Fragmento 3 - Código para a segunda página

```
#Segunda Página
def second_frame(self):
    #Lógica para fechar a primeira página e abrir a segunda
    self.initial_frame.pack_forget()
    self.page2_frame = Frame(self.window, bg="#222222", width=400, height=550)
    self.page2_frame.pack_propagate(False)
    self.Poppins = font.Font(family="Poppins", size=12)

    #Título
    title_label = Label(self.page2_frame, text="CREATE YOUR \n PLAYLIST",
font=("Poppins", 25, "bold"), bg="#222222", fg="#1ED760")
    title_label.pack(pady=50)

    #Configuração dos Campos de Entrada de Dados
    entry_frame = Frame(self.page2_frame, bg="#222222")
    entry_frame.pack()

    #Campo - Artists
    self.artists_label = Label(entry_frame, text="Artists/Bands (max. 50):",
font=self.Poppins, bg="#222222", fg="white")
    self.artists_label.grid(row=1, column=0, padx=5, pady=5, sticky="w")
    self.artists_entry = Entry(entry_frame, font=self.Poppins, bg="#4E4E4E")
    self.artists_entry.grid(row=2, column=0, padx=5, pady=12, sticky="ew")

    #Campo - Num of Songs
    self.num_songs_label = Label(entry_frame, text="Number of Songs (max.
100):", font=self.Poppins, bg="#222222", fg="white")
    self.num_songs_label.grid(row=3, column=0, padx=5, pady=5, sticky="w")
    self.num_songs_entry = Entry(entry_frame, font=self.Poppins, bg="#4E4E4E")
    self.num_songs_entry.grid(row=4, column=0, padx=5, pady=12, sticky="ew")

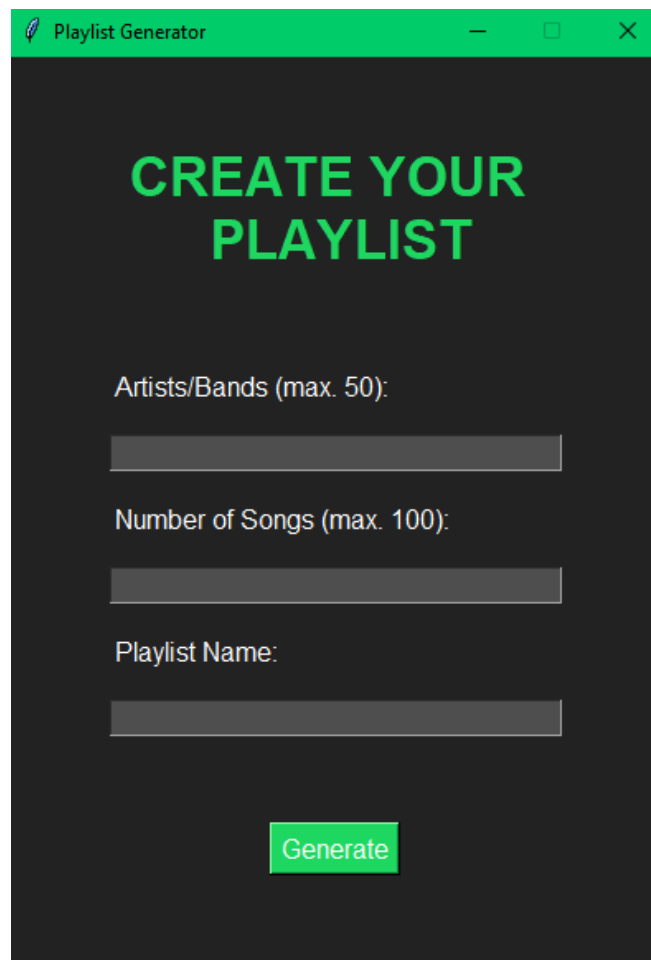
    #Campo - Playlist Name
    self.playlist_name_label = Label(entry_frame, text="Playlist Name:",
font=self.Poppins, bg="#222222", fg="white")
    self.playlist_name_label.grid(row=5, column=0, padx=5, pady=5, sticky="w")
    self.playlist_name_entry = Entry(entry_frame, font=self.Poppins, bg="#4E4E4E",
width=30)
    self.playlist_name_entry.grid(row=6, column=0, padx=5, pady=12, sticky="ew")

    #Botão "Generate" que ativa a função de geração de playlist
    generate_button = Button(self.page2_frame, text="Generate", font=self.Poppins,
bg="#1ED760", fg="white", command=self.generate_playlist)
    generate_button.pack(pady=40)

    self.page2_frame.pack()
```

Após pressionar “Continue”, a segunda página será como a descrita na Figura 2, visto a implementação do código. Note que é necessário que os artistas sejam separados por vírgula e que o máximo é de 50, e com um máximo de 100 músicas. Esses parâmetros foram escolhas do programador, e, portanto, podem ser expandidos se necessário.

Figura 2 - Página 2



The image shows a web browser window titled "Playlist Generator". The main heading is "CREATE YOUR PLAYLIST" in large, bold, green capital letters. Below this, there are three input fields with labels: "Artists/Bands (max. 50):", "Number of Songs (max. 100):", and "Playlist Name:". Each label is followed by a grey rectangular input box. At the bottom center, there is a green rectangular button with the word "Generate" in white text.

4.4. Def generate_playlist

Como visto, após clicar em “Generate”, a playlist será gerada de acordo com os parâmetros digitados. Neste projeto, optou-se por realizar uma pesquisa por álbuns relacionados ao artista — no *Spotify*, especificamente, cada um é relacionado a um “URI” —, para que possa-se selecionar músicas aleatórias do mesmo.

Todavia, antes disso, é necessário se certificar de que o *Spotify* irá permitir a utilização do programa. Sendo assim, é necessário realizar uma autenticação com sua conta. Para isso, fez-se uso do *Spotify Developer*, uma aplicação própria da empresa para que desenvolvedores possam criar suas aplicações. No site, é necessário que o usuário crie seu próprio projeto e, assim, irá receber um “client id” e um “client secret”. Também no site, é necessário informar o *local host* em que o usuário será direcionado para a aplicação — <http://localhost:8888/callback>, por exemplo. Tendo essas credenciais, juntamente ao nome de usuário associado ao perfil, é possível rodar a aplicação.

Para receber os dados digitados nos campos de entrada da função anterior, utilizou-se o método `.get()`. Assim, cria-se uma lista com todos os artistas e que posteriormente será percorrida para que as faixas sejam adicionadas na playlist. Além disso, é necessário calcular quantas faixas serão adicionadas por artistas:

- Se $(\text{num_songs}) \bmod (\text{len}(\text{artists})) = 0$, a distribuição será igualitária;
- Se $(\text{num_songs}) \bmod (\text{len}(\text{artists})) > 0$, os primeiros artistas digitados receberão músicas adicionais.

Desse modo, destaca-se o fragmento de código que cria essa lógica.

Fragmento 4 - Código para a geração da Playlist

```
#Geração da Playlist
def generate_playlist(self):
    #Lógica para conseguir o Token de Acesso do Spotify
    client_id = 'SEU-CLIENT-ID'
    client_secret = 'SEU-SECRET-ID'
    redirect_uri = 'SEU-LOCAL-HOST'
    credentials = oauth2.SpotifyOAuth(client_id, client_secret, redirect_uri)
    #Sistema de autenticação para permissão do programa
    sp = spotipy.Spotify(auth_manager=credentials)
    #Coleta dos dados preenchidos nas caixas da página 2
    artists = self.artists_entry.get().split(',')
    num_songs = int(self.num_songs_entry.get())
    playlist_name = self.playlist_name_entry.get()
    songs_per_artist = num_songs // len(artists)
    #Cálculo do resto da divisão para caso a quantidade de artistas não seja divisível pelo num de
    músicas
    remainder = num_songs % len(artists)
    #Restrições da quantidade e artistas e número de músicas (é possível alterar):
    if len(artists) <= 50 and num_songs <= 100:
        #Criação da Playlist
        #OBS: É necessário preencher com seu próprio usuário do Spotify.
        playlist = sp.user_playlist_create(user='SEU-USERNAME', name=playlist_name)
        #Lógica de busca pelos artistas no Spotify (seleciona o primeiro compatível com o nome
        digitado)
        for artist_name in artists:
            results = sp.search(q=artist_name, limit=1, type='artist')
            artist_uri = results['artists']['items'][0]['uri']
            #Vetor com todas as músicas do artista
            tracks = []
            #Todos os albums do artista:
            albums = sp.artist_albums(artist_uri, album_type='album')
            #Adiciona-se cada faixa de cada album na lista tracks:
            for album in albums['items']:
                album_tracks = sp.album_tracks(album['id'])
                tracks.extend(track['uri'] for track in album_tracks['items'])
```

```

        #Embaralhamento para que a ordem seja aleatória
        random.shuffle(tracks)
        #Caso o resto seja diferente de 0, serão adicionadas mais músicas aos primeiros
    artistas/bandas
        additional = 0
        if remainder > 0:
            additional = 1
            remainder -= 1
        if additional > 0:
            sp.playlist_add_items(playlist['id'], tracks[:songs_per_artist + additional])
        else:
            sp.playlist_add_items(playlist['id'], tracks[:songs_per_artist])
    #Link da Playlist:
    self.playlist_url = f"https://open.spotify.com/playlist/{playlist['id']}"
    print("Open the Playlist:", self.playlist_url)
    #Caso as restrições não sejam atendidas:
    else:
        if len(artistas) > 50:
            print("Adicione uma quantidade de artistas menor ou igual que 50.")
        if num_songs > 100:
            print("Adicione uma quantidade de artistas menor ou igual que 100.")
    #Depois de gerada, abre-se a terceira página
    self.third_frame()

```

4.5. Def thid_frame e open_playlist

Estas últimas funções são utilizadas para, respectivamente, fornecer um botão com um link direto para a playlist e para, de fato, abri-la no navegador padrão (utilizando a biblioteca webbrowser).

Fragmento 5 - Terceira Página

```

#Terceira Página
def third_frame(self):
    # Lógica para fechar a segunda página e abrir a terceira
    self.page2_frame.pack_forget()
    self.page3_frame = Frame(self.window, bg="#222222", width=400, height=550)
    self.page3_frame.pack_propagate(False)
    self.page3_frame.pack(fill="both")
    self.Poppins = font.Font(family="Poppins", size=12)
    #Título
    title = Label(self.page3_frame, text="PLAYLIST \n GENERATED!", font=(self.Poppins, 25,
"bold"),bg="#222222", fg="#1ED760")
    title.pack(pady=50)
    #Botão para abrir a playlist (Ativa a def open_playlist)
    open_button = Button(self.page3_frame, text="Open the Playlist", font=self.Poppins,
bg="#1ED760", fg="white",command=self.open_playlist)
    open_button.pack(pady=80)
    #Mensagem de créditos
    message_label = Label(self.page3_frame, text="© Programa criado por \n Leonardo de Sousa
Marques para a \n Disciplina INE5401, com auxílio de \n Spotify Developer", font=(self.Poppins,
12), bg="#222222", fg="white")

```

```

message_label.pack(pady=40)
self.page3_frame.pack()
#Função para abrir a playlist no navegador padrão:
def open_playlist(self):
    webbrowser.open(self.playlist_url)

```

5. EXEMPLO DE FUNCIONAMENTO

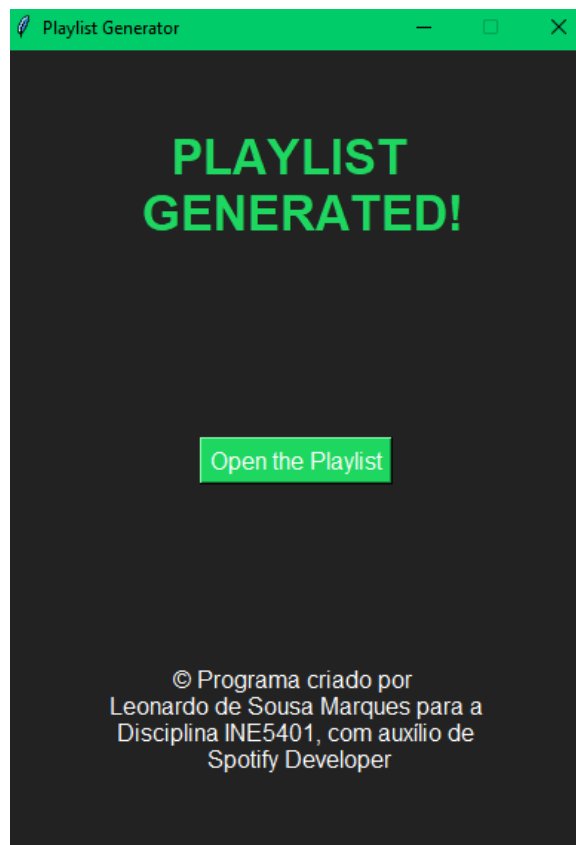
Tendo visto a estruturação do código, agora pode-se apresentar um exemplo da aplicação em funcionamento. Imagine que queira-se criar uma playlist com as bandas *Queen*, *Nirvana* e *The Rolling Stones*, com 32 músicas e nome “Playlist de Rock - 01”.

- **Passo 1:** rodar o código no editor de escolha (VSCode...) ou através do terminal (python PlaylistGenerator.py). O programa irá abrir conforme visto na Figura 1.
- **Passo 2:** digitar os dados em suas respectivas caixas.

Figura 3 - Exemplo (1)

- **Passo 3:** pressionar “*Generate*” e aguardar que a página 3 seja aberta com o botão de link direto à playlist.

Figura 4 - Exemplo (2)



- **Passo 4:** pressionar “*Open the Playlist*”. Conforme solicitado, foram adicionadas 32 músicas — 11 para *Queen*, 11 para *Nirvana* e 10 para *The Rolling Stones*.

Figura 5 - Screenshot da playlist no Spotify



6. REFERÊNCIAS

- [1] LABAK, Josué. Introdução a Python - Tkinter. UNESP. Disponível em: <www.dcc.ufrj.br/~fabiom/mab225/tutorialtkinter.pdf>. Acesso em 20 de out de 2023.
- [2] Spotipy. Spotipy Documentation. Disponível em: <spotipy.readthedocs.io/en/2.22.1/>. Acesso em 22 de out de 2023.
- [3] Spotify. Spotify Documentation - Web API. Disponível em: <developer.spotify.com/documentation/web-api>. Acesso em 22 de out de 2023.
- [4] Synsation. Spotify OAuth: Automating Discover Weekly Playlist - Full Tutorial. Youtube, 2023. Disponível em: <www.youtube.com/watch?v=mBycigbJQzA>. Acesso em 25 de out de 2023.