Aplikacja do porównywania artystów zarejestrowanych w serwisie Spotify

PROJEKT ZALICZENIOWY Z PRZEDMIOTU

SKŁADOWANIE DANYCH W SYSTEMACH BIG DATA

MAKAREWICZ AGATA 298827

WIŚNIEWSKI JACEK 298849

17.01.2022r.

Spis treści

1 Cel projektu

Celem projektu jest udostępnienie artystom oraz miłośnikom muzyki raportu, który porównywałby wyniki muzyków na platformie Spotify. W tym celu powstała aplikacja kliencka, która służy do generowania widoków ze statystykami artystów. Ponieważ serwis Spotify udostępnia dane ponad miliona artystów przez API internetowe, konieczne jest wykorzystanie architektury dedykowanej dla danych wielkoskalowych (*ang. Big Data*) dla sprawnego działania aplikacji.

2 Zbiory danych

Projekt bazuje na 3 zbiorach danych, przy czym dane te pozyskane zostały na dwa sposoby. Po pierwsze, ze strony *https://kworb.net/spotify/artists.html* przy użyciu skryptu języka Python (*scraping.py*) pobrane zostały identyfikatory 5000 najpopularniejszych artystów według rankingu bazującego na danych udostępnianych przez Spotify (*https://www.spotifycharts.com*). Następnie uzyskane identyfikatory zostały podane jako parametry w zapytaniach kierowanych bezpośrednio do Web API Spotify, w wyniku których otrzymaliśmy dane o artystach. Dane te udostępniane są w formacie JSON, o poniższym schemacie.

```
root
 |-- artists: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- external_urls: struct (nullable = true)
 |    |    |    |-- spotify: string (nullable = true)
 |    |    |-- followers: struct (nullable = true)
 |    |    |    |-- href: string (nullable = true)
 |    |    |    |-- total: long (nullable = true)
 |    |    |-- genres: array (nullable = true)
 |    |    |    |-- element: string (containsNull = true)
 |    |    |-- href: string (nullable = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- images: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- height: long (nullable = true)
 |    |    |    |    |-- url: string (nullable = true)
 |    |    |    |    |-- width: long (nullable = true)
 |    |    |-- name: string (nullable = true)
 |    |    |-- popularity: long (nullable = true)
 |    |    |-- type: string (nullable = true)
 |    |    |-- uri: string (nullable = true)
```

Figura 1: Schemat danych dotyczących artystów.

Takich plików jest 125, każdy zawiera informacje o 40 artystach, co w sumie daje nam 5000 „rekordów" z różnego rodzaju zagnieżdżonymi strukturami.

Drugim sposobem pozyskania danych było skorzystanie z pakietu języka Python *spotipy*, który umożliwia korzystanie z Web API Spotify. Skrypt z jego użyciem (*spotipy.ipynb*) posłużył do pobrania listy albumów oraz 10 najpopularniejszych utworów pozyskanych wcześniej artystów. Otrzymaliśmy 5000 plików dotyczących albumów oraz 5000 dotyczących utworów, w formacie JSON, o poniższych schematach.

```
root
 |-- href: string (nullable = true)
 |-- items: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- album_group: string (nullable = true)
 |    |    |-- album_type: string (nullable = true)
 |    |    |-- artists: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- external_urls: struct (nullable = true)
 |    |    |    |    |    |-- spotify: string (nullable = true)
 |    |    |    |    |-- href: string (nullable = true)
 |    |    |    |    |-- id: string (nullable = true)
 |    |    |    |    |-- name: string (nullable = true)
 |    |    |    |    |-- type: string (nullable = true)
 |    |    |    |    |-- uri: string (nullable = true)
 |    |    |-- available_markets: array (nullable = true)
 |    |    |    |-- element: string (containsNull = true)
 |    |    |-- external_urls: struct (nullable = true)
 |    |    |    |-- spotify: string (nullable = true)
 |    |    |-- href: string (nullable = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- images: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- height: long (nullable = true)
 |    |    |    |    |-- url: string (nullable = true)
 |    |    |    |    |-- width: long (nullable = true)
 |    |    |-- name: string (nullable = true)
 |    |    |-- release_date: string (nullable = true)
 |    |    |-- release_date_precision: string (nullable = true)
 |    |    |-- total_tracks: long (nullable = true)
 |    |    |-- type: string (nullable = true)
 |    |    |-- uri: string (nullable = true)
 |-- limit: long (nullable = true)
 |-- next: string (nullable = true)
 |-- offset: long (nullable = true)
 |-- previous: string (nullable = true)
 |-- total: long (nullable = true)
```

```
root
 |-- tracks: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- album: struct (nullable = true)
 |    |    |    |-- album_type: string (nullable = true)
 |    |    |    |-- artists: array (nullable = true)
 |    |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |    |-- external_urls: struct (nullable = true)
 |    |    |    |    |    |    |-- spotify: string (nullable = true)
 |    |    |    |    |    |-- href: string (nullable = true)
 |    |    |    |    |    |-- id: string (nullable = true)
 |    |    |    |    |    |-- name: string (nullable = true)
 |    |    |    |    |    |-- type: string (nullable = true)
 |    |    |    |    |    |-- uri: string (nullable = true)
 |    |    |    |-- external_urls: struct (nullable = true)
 |    |    |    |    |-- spotify: string (nullable = true)
 |    |    |    |-- href: string (nullable = true)
 |    |    |    |-- id: string (nullable = true)
 |    |    |    |-- images: array (nullable = true)
 |    |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |    |-- height: long (nullable = true)
 |    |    |    |    |    |-- url: string (nullable = true)
 |    |    |    |    |    |-- width: long (nullable = true)
 |    |    |    |-- name: string (nullable = true)
 |    |    |    |-- release_date: string (nullable = true)
 |    |    |    |-- release_date_precision: string (nullable = true)
 |    |    |    |-- total_tracks: long (nullable = true)
 |    |    |    |-- type: string (nullable = true)
 |    |    |    |-- uri: string (nullable = true)
 |    |    |-- artists: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- external_urls: struct (nullable = true)
 |    |    |    |    |    |-- spotify: string (nullable = true)
 |    |    |    |    |-- href: string (nullable = true)
 |    |    |    |    |-- id: string (nullable = true)
 |    |    |    |    |-- name: string (nullable = true)
 |    |    |    |    |-- type: string (nullable = true)
 |    |    |    |    |-- uri: string (nullable = true)
 |    |    |-- disc_number: long (nullable = true)
 |    |    |-- duration_ms: long (nullable = true)
 |    |    |-- explicit: boolean (nullable = true)
 |    |    |-- external_ids: struct (nullable = true)
 |    |    |    |-- isrc: string (nullable = true)
 |    |    |-- external_urls: struct (nullable = true)
 |    |    |    |-- spotify: string (nullable = true)
 |    |    |-- href: string (nullable = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- is_local: boolean (nullable = true)
 |    |    |-- is_playable: boolean (nullable = true)
 |    |    |-- name: string (nullable = true)
 |    |    |-- popularity: long (nullable = true)
 |    |    |-- preview_url: string (nullable = true)
 |    |    |-- restrictions: struct (nullable = true)
 |    |    |    |-- reason: string (nullable = true)
 |    |    |-- track_number: long (nullable = true)
 |    |    |-- type: string (nullable = true)
 |    |    |-- uri: string (nullable = true)
```

Figura 2: Schematy danych dotyczących albumów (po lewej) oraz najpopularniejszych utworów (po prawej) danego artysty.

Wszystkie dane zostały pobrane jednokrotnie, ponieważ token dostępu do Web Api Spotify musi być ręcznie generowany co godzinę.

3 Architektura systemu

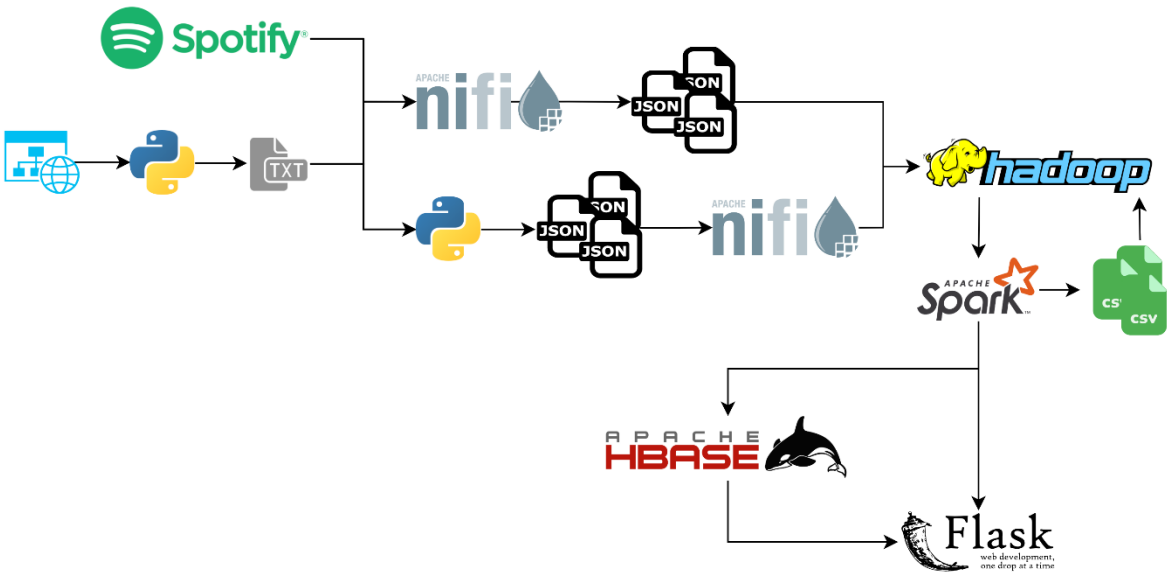


Figura 3: Diagram architektury rozwiązania.

Na początku przy pomocy skryptu języka Python ze strony internetowej pobrano do pliku tekstowego identyfikatory artystów z serwisu Spotify. Następnie dane pozyskano na dwa sposoby. Pierwszy wariant to wykorzystanie Apache NiFi do wykonania zapytania do Web API Spotify, w wyniku czego otrzymaliśmy pliki w formacie JSON dotyczące artystów. Drugi wariant to wykorzystanie skryptu języka Python do wykonania analogicznych zapytań do Web API Spotify celem uzyskania plików dotyczących albumów oraz utworów (również w formacie JSON). Pliki te po pobraniu załadowano do Apache NiFi. Wszystkie wspomniane pliki utworzyły zbiór referencyjny składowany w systemie HDFS (ładowanie przy pomocy Apache NiFi). Następnie przy pomocy Apache Spark pozyskane dane zostały wstępnie przetworzone oraz złączone w jeden zbiór, który załadowano w postaci plików CSV do HDFS. Na ich podstawie, ponownie przy użyciu Apache Spark, generowane są widoki wsadowe, które składowane są w Apache HBase. Proces ten odbywa się poprzez aplikację Flask, która umożliwia użytkownikowi wyświetlenie wykonanych analiz.

4 Opis rozwiązania

4.1 Pozyskiwanie, przetwarzanie i składowanie danych

Pozyskiwanie danych rozpoczęło pobranie ze strony *https://kworb.net/spotify/artists.html* identyfikatorów 5000 najpopularniejszych artystów w serwisie według rankingu bazującego na danych udostępnianych przez Spotify (*https://www.spotifycharts.com*). Posłużył do tego skrypt języka Python (*scraping.py*), wykorzystujący bibliotekę *BeautifulSoup*. Krok ten był konieczny do utworzenia referencyjnego zbioru danych, ponieważ Web API Spotify nie umożliwia pobrania danych dla wszystkich artystów lub pewnej ich części; wymagane jest podanie konkretnych identyfikatorów.

Po utworzeniu wspomnianego pliku zawierającego identyfikatory artystów, pozyskane zostały dane na temat ich albumów oraz 10 najpopularniejszych utworów. W tym celu ponownie wykorzystano skrypt języka Python (*spotipy.ipynb)*; dane pobrano przy użyciu biblioteki *spotipy* umożliwiającej wykonanie zapytań do Web API Spotify. Rozwiązanie to wybrano z uwagi na znacznie krótszy czas pozyskiwania danych niż w przypadku bezpośredniego zapytania (z uwagi na ograniczenie liczby elementów, które mogą być w nim zawarte). Skrypt wymaga autoryzacji poprzez podanie tokenu wygenerowanego na stronie https://developer.spotify.com/. W kolejnym kroku wczytano identyfikatory artystów, po czym w pętli pobrano potrzebne informacje (w postaci plików w formacie JSON) i umieszczono w dedykowanych folderach (*/albums*, */tracks*), z identyfikatorem artysty jako nazwą.

Następnym etapem pozyskiwania danych było utworzenie odpowiedniego przepływu w Apache NiFI celem pobrania informacji dotyczących samych artystów.

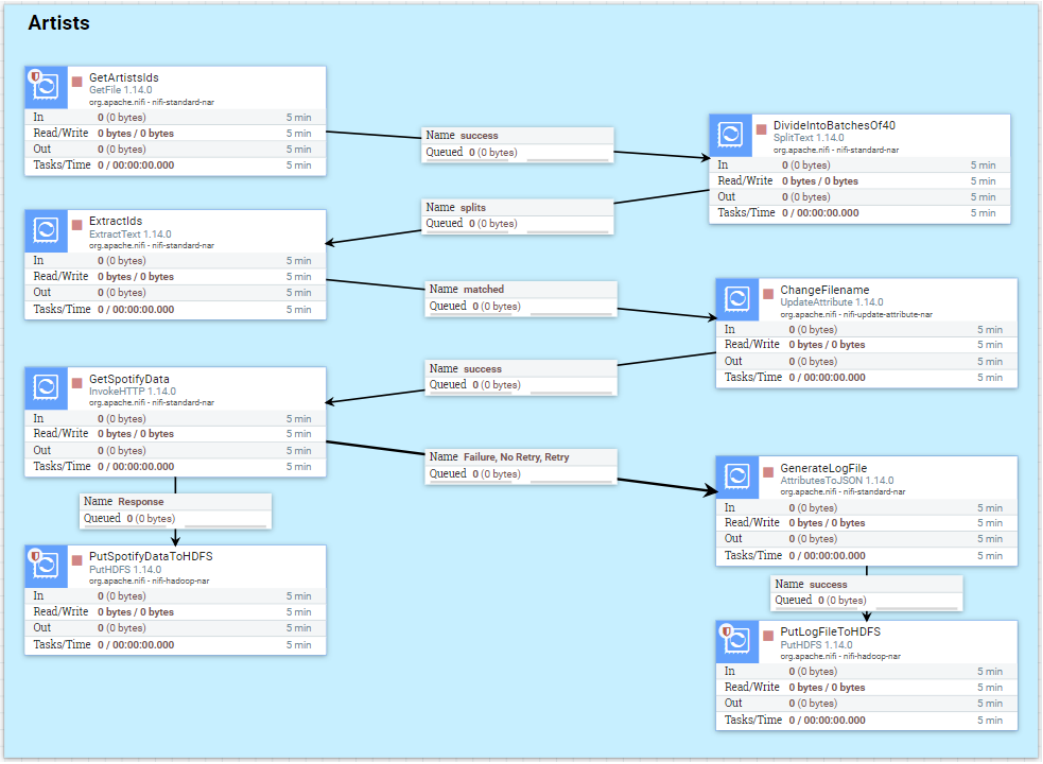


Figura 4: Przepływ danych dotyczący danych o artystach.

Na początku do przepływu przy pomocy procesora *GetFile* pobierany jest wspomniany wcześniej plik tekstowy z identyfikatorami artystów. Plik zawiera 5000 identyfikatorów, po 40 w jednej linii. Taki podział spowodowany jest ograniczeniami Web API Spotify – w jednym zapytaniu nie można podać więcej elementów. Przy pomocy procesora *SplitText* plik dzielony jest na 125 plików przepływu (po 40 identyfikatorów w każdym). Następnie procesor *ExtractText* wydobywa zawartość każdego z plików i przypisuje ją do nich jako atrybut *content*. W kolejnym kroku przy pomocy procesora *UpdateAttribute* zmieniane są nazwy plików według wzorca:

${filename:substringBefore("_")}_${fragment.index}.json

W rezultacie otrzymywane są pliki *artists_1*, … *artists_125*. Po wykonaniu powyższych operacji pliki przepływu trafiają do procesora *InvokeHTTP*, przy pomocy którego wykonywane jest zapytanie do Web API Spotify, z zawartością plików przepływu (identyfikatorami artystów) jako parametrem:

https://api.spotify.com/v1/artists?ids=${content}

Do wykonania zapytania konieczne jest dodanie dwóch właściwości do procesora (opcji wywołania zapytania do API):

- Accept : application/json
- Authorization : Bearer <token>

Token umożliwiający dostęp do API trzeba generować co godzinę. Rezultatem są pliki w formacie JSON zawierające informacje o poszczególnych artystach, takie jak liczba obserwatorów, gatunki, w których tworzą, czy popularność. Pliki te ładowane są do dedykowanego folderu w HDFS (*/spotify/artists*) przy pomocy procesora *PutHDFS*.

Dodatkowo do przepływu dołączone zostały dwa procesory – *AttributesToJSON* oraz *PutHDFS* – odpowiedzialne za obsługę błędów. Pierwszy z nich wydobywa atrybuty pliku przepływu takie jak URL, odpowiedź HTTP wraz z kodem czy indeks danego pliku przepływu, do pliku w formacie JSON, a drugi ładuje wygenerowane pliki do HDFS (*/spotify/logs*).

W Apache NiFi utworzone zostały jeszcze dwa przepływy, dla wcześniej wspomnianych danych dotyczących albumów oraz utworów. Pozyskane uprzednio pliki w formacie JSON ładowane są do odpowiednich folderów w HDFS (*/spotify/albums, /spotify/tracks*).

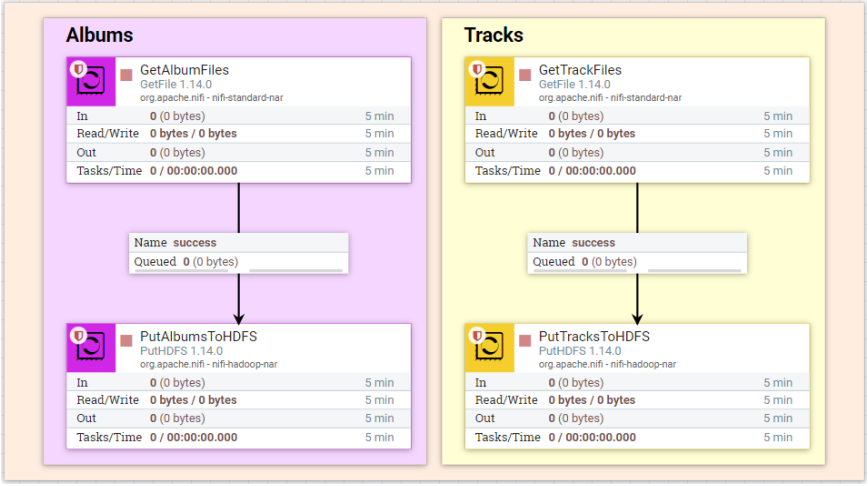


Figura 5: Przepływy danych dotyczące danych o albumach i utworach.

W następnym kroku dane są przetwarzane przy pomocy skryptu języka Python wykorzystującego *PySpark* – interfejs dla Apache Spark w Pythonie. Z plików z danymi o artystach wybrane zostały informacje o ich identyfikatorze, imieniu i nazwisku bądź pseudonimie, liczbie obserwujących, popularności, oraz gatunkach, w których tworzą. Następnie usunięto zduplikowane rekordy – po tej operacji w zbiorze pozostało 4989 unikalnych artystów. Z plików z danymi o albumach wybrane zostały informacje o identyfikatorze artysty (na postawie adresu URL), rodzaju albumu, jego identyfikatorze, tytule, dacie wydania oraz liczbie utworów. Wyfiltrowano dane według rodzaju albumu – w zbiorze pozostały jedynie „klasyczne" albumy (nie kompilacje) oraz single. Oprócz tego dokonano drobnych modyfikacji – dla 3 artystów zmieniono identyfikatory. Wynikało to z faktu, iż dla niektórych artystów istnieją dwa działające identyfikatory; w tym wypadku w zbiorze danych o artystach trzech z nich miało przypisany jeden identyfikator, a w zbiorze danych o albumach – drugi, co powodowało powstawanie niepotrzebnych braków danych przy łączeniu zbiorów. Te same modyfikacje zostały przeprowadzone w zbiorze danych o utworach. Z tego zbioru wybrano informacje o identyfikatorze albumu, długości utworu, jego identyfikatorze, tytule, popularności oraz o tym, czy zawiera on wulgaryzmy. Dodatkowo dodano informację o identyfikatorze artysty na podstawie nazwy pliku. Wszystkie trzy zbiory zostały połączone na podstawie identyfikatorów artystów oraz albumów oraz załadowane do HDFS w postaci plików w formacie CSV.

Do składowania danych wykorzystany został również projekt Apache HBase. Ładowane są tam widoki wsadowe tworzone przy pomocy Apache Spark poprzez interakcję ze stworzoną aplikacją Flask. Tabela w Apache HBase służy do przechowywania wszystkich 3 rodzajów tworzonych widoków. W jej obrębie wyróżnione zostały 3 rodziny kolumn – dotyczące danych o artystach, albumach oraz utworach, gdyż po takie podzbiory zmiennych sięgamy poprzez aplikację.

4.2 Analiza danych (generowanie widoków wsadowych)

W trakcie przetwarzania danych przy użyciu Apache Spark wygenerowano kilka widoków wsadowych, które pozwoliły na lepsze zrozumienie pozyskanych danych oraz wydobycie interesujących informacji.

track_name	artist_name	track_popularity
abcdefu	GAYLE	99
All I Want for Christmas Is You	Mariah Carey	99
Enemy (with JID) - from the series Arcane League of Legends	JID	98
Enemy (with JID) - from the series Arcane League of Legends	Imagine Dragons	98
Woman	Doja Cat	97
good 4 u	Olivia Rodrigo	96
Shivers	Ed Sheeran	96
Do It To It	ACRAZE	96
Bad Habits	Ed Sheeran	96
All Too Well (10 Minute Version) (Taylor's Version) (From The Vault)	Taylor Swift	96

Tabela 1: Dziesięć najpopularniejszych utworów w całym zbiorze wraz z imieniem i nazwiskiem/pseudonimem artysty.

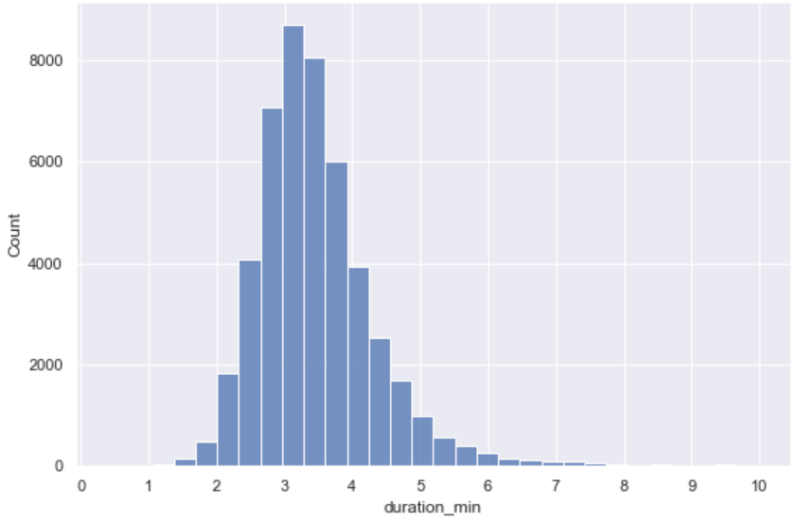


Figura 6: Rozkład zmiennej opisującej długość trwania utworu (po konwersji z milisekund na minuty) z pominięciem utworów dłuższych niż 10 min (takich utworów było tylko 125 i znacznie zaburzały rozkład).

genres	count
pop	363
dance pop	291
latin	186
trap latino	186
pop rap	180
rap	177
funk carioca	161
tropical house	157
pop dance	144
trap	136

genres	avg_pop
barbadian pop	90.0
british invasion	89.0
merseybeat	89.0
deep underground hip hop	88.0
new orleans rap	88.0
speedrun	87.0
pittsburgh rap	86.5
ohio hip hop	85.5
funk rock	84.0
old school thrash	84.0

Tabela 2, 3: Dziesięć najczęściej występujących gatunków (według liczby artystów w nich tworzących) oraz dziesięć gatunków o najwyższej średniej popularności artystów w nich tworzących (w przypadku barbadian pop jedyną artystką jest Rihanna).

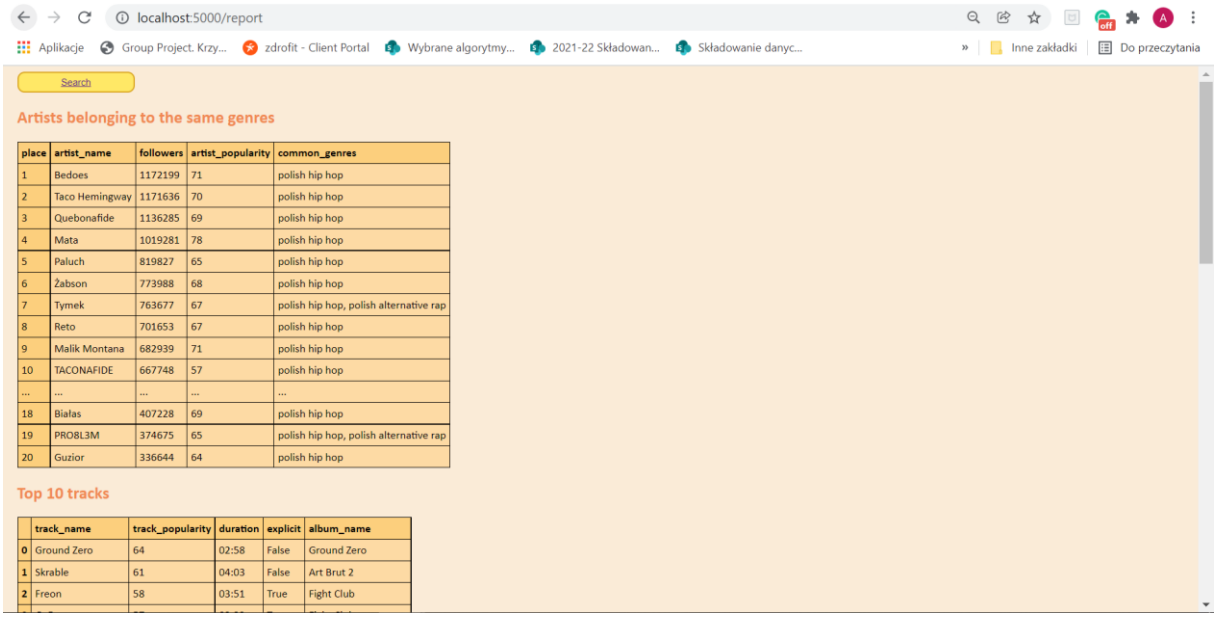
Na podstawie przetworzonych wcześniej danych zostały utworzone także widoki wsadowe, które znalazły się w warstwie prezentacyjnej. Generowanie widoków odbywa się przy użyciu funkcji napisanych w języku Python, wykorzystujących *PySpark* do skorzystania z Apache Spark oraz pakiet *happybase* do połączenia z Apache HBase, przyjmujących na wejściu identyfikator artysty. Utworzono następujące widoki:

- Porównanie liczby obserwatorów oraz rankingu popularności Spotify dla artystów, tworzących muzykę w podobnych gatunkach co artysta o podanym identyfikatorze
- Informacje na temat albumów wydanych przez wybranego artystę, posortowane według daty wydania
- Informacje na temat najpopularniejszych piosenek artysty, posortowane według popularności

Każda z funkcji wywołana po raz pierwszy dla wybranego identyfikatora artysty tworzy widok wsadowy przy pomocy Apache Spark (PySpark), a następnie ładuje go do tabeli w Apache HBase oraz zwraca jako wynik. Jeżeli funkcja zostanie wywołana po raz kolejny dla tego samego artysty, widok wsadowy zostanie wczytany z tabeli w Apache HBase.

4.3 Warstwa prezentacyjna

Warstwa prezentacyjna rozwiązania przygotowana została w postaci aplikacji Flask. Składa się z dwóch widoków – *search* oraz *report*. Pierwszy z nich zawiera formularz z jednym polem tekstowym, w którym użytkownik może podać identyfikator artysty. Po przesłaniu formularza generowany jest drugi z widoków, który zawiera raport przygotowany przy pomocy PySpark'a. Dla wybranego artysty wyświetlana jest lista artystów o największej liczbie obserwujących, tworzących w tych samych gatunkach, 10 najpopularniejszych utworów, oraz lista albumów.



place	artist_name	followers	artist_popularity	common_genres
1	Bedoes	1172199	71	polish hip hop
2	Taco Hemingway	1171636	70	polish hip hop
3	Quebonafide	1136285	69	polish hip hop
4	Mata	1019281	78	polish hip hop
5	Paluch	819827	65	polish hip hop
6	Żabson	773988	68	polish hip hop
7	Tymek	763677	67	polish hip hop, polish alternative rap
8	Reto	701653	67	polish hip hop
9	Malik Montana	682939	71	polish hip hop
10	TACONAFIDE	667748	57	polish hip hop
...	...	...	...	...
18	Białas	407228	69	polish hip hop
19	PRO8L3M	374675	65	polish hip hop, polish alternative rap
20	Guzior	336644	64	polish hip hop

	track_name	track_popularity	duration	explicit	album_name
0	Ground Zero	64	02:58	False	Ground Zero
1	Skrable	61	04:03	False	Art Brut 2
2	Freon	58	03:51	True	Fight Club

Figura 7: Fragment raportu zawartego w warstwie prezentacyjnej.

5 Testowanie rozwiązania

W tym rozdziale zostały zaprezentowane testy funkcjonalne obejmujące zaimplementowane i skonfigurowane komponenty.

Nr	Cel	Podjęte kroki	Oczekiwany wynik
1	Otrzymanie listy ID najpopularniejszych 5000 artystów według Spotify	Uruchomiony został skrypt *scraping.py*, który pobiera ze strony *https://kworb.net/spotify/artists.html* identyfikatory pierwszych 5000 artystów.	Plik tekstowy zawierający 5000 ID, oddzielonych przecinkami, po 40 w jednej linii.
2	Załadowanie informacji na temat artystów do HDFS	Uruchomiony został przepływ danych (dedykowany danym o	Folder *artists* w HDFS zawiera 125 plików w

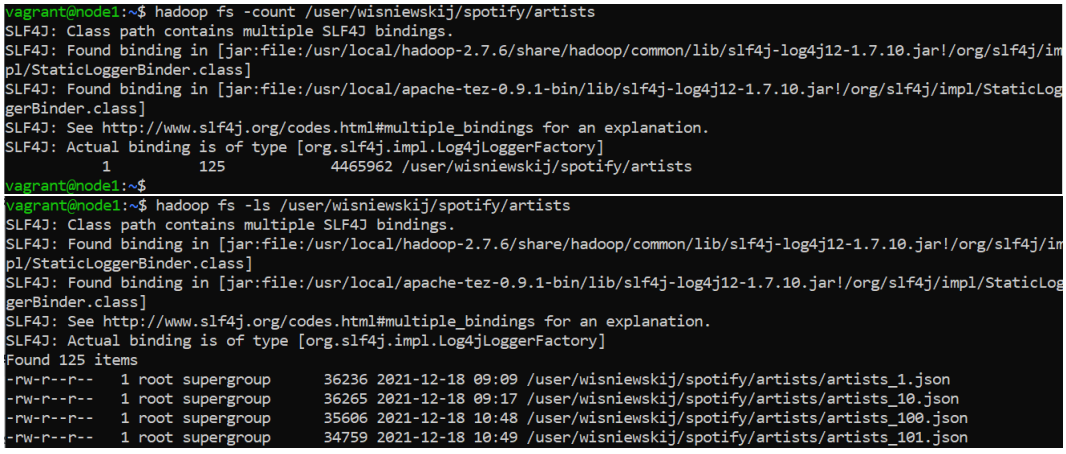
		artystach) utworzony w Apache NiFi.	formacie JSON, o nazwach artists_* .
3	Otrzymanie informacji na temat albumów najpopularniejszych 10 artystów według Spotify	Uruchomiony został skrypt *spotipy_albums.ipynb* z wygenerowanym na stronie *developer.spotify.com* nowym kluczem autoryzującym.	Plik w formacie JSON zawierający informacje na temat albumów 10 artystów.
4	Otrzymanie informacji na temat piosenek najpopularniejszych 10 artystów według Spotify	Uruchomiony został skrypt *spotipy_tracks.ipynb* z wygenerowanym na stronie *developer.spotify.com* nowym kluczem autoryzującym.	Plik w formacie JSON zawierający informacje na temat piosenek 10 artystów.
5	Utworzenie tabeli zawierającej komplet informacji otrzymanych w poprzednich krokach	Uruchomiony został skrypt *PySparkDataPreprocessing.ipynb*, w którym dane dotyczące artystów, albumów oraz utworów są transformowane i łączone w jedną tabelę.	Na końcu skryptu powinna pojawić się tabela zawierająca dane dotyczące artystów, albumów i utworów.
6	Zapisanie tabeli z kompletem informacji do HDFS	Potrzebne kroki wykonane przy okazji poprzedniego testu (na końcu wspomnianego skryptu).	Folder *data* w HDFS zawiera pliki CSV o schemacie odpowiadającym stworzonej tabeli.
7	Utworzenie raportu dla artysty o ID 4O15NlyKLIASxsJ0PrXPfz	Uruchomiony został skrypt */usr/local/hbase/bin/hbase thrift* na wirtualnej maszynie, a następnie został uruchomiony skrypt *PySpark_report_test.ipynb* z podanym ID jako parametrem.	Na końcu skryptu powinny pojawić się 3 tabele utworzone przy pomocy funkcji języka Python z wykorzystaniem pakietu PySpark.
8	Ponowne utworzenie raportu dla artysty o ID 4O15NlyKLIASxsJ0PrXPfz (w celu przetestowania wczytywania raportu z Apache HBase)	Zostały powtórzone kroki z poprzedniego testu.	Skrypt powinien zwrócić te same wyniki co poprzedni test, ale w znacznie krótszym czasie.
9	Utworzenie raportu dla artysty o ID 1SKeSGQ3LMHYCEgqFGvJbE (Krzysztof Krawczyk)	Zostały powtórzone kroki z poprzednich testów, ze zmienionym wyłącznie ID artysty.	Ponieważ artysta o podanym ID nie należy do grona 5000 najpopularniejszych artystów według wykorzystywanego rankingu, powinny zostać wyświetlone wartości „-1”.
10	Uruchomienie aplikacji Flask	W terminalu, po przejściu do folderu *spotify*, wywołane zostały kolejno komendy: *export FLASK_APP=app* *python3 -m flask run --host=0.0.0.0*	W przeglądarce pod adresem *localhost:5000/search* wyświetla się strona z polem formularza.

11	Stworzenie raportu w aplikacji Flask dla artysty o ID 7v49oVVUhvIQG5EK0jkcF7 (Probl3m)	Przy wykonanych krokach z poprzedniego testu, w przeglądarce pod adresem *localhost:5000/search* w polu formularza wpisano podany ID artysty.	W przeglądarce pod adresem *localhost:5000/report* wyświetla się strona z raportem dla danego artysty (3 tabele).
12	Stworzenie raportu w aplikacji Flask dla artysty o ID 72T7x96EAqN2UWvAgobYfv (Sizzla) (artysta nie jest w zbiorze artystów)	Zostały powtórzone kroki z poprzedniego testu, ze zmienionym wyłącznie ID artysty.	W przeglądarce pod adresem *localhost:5000/report* wyświetla się komunikat o tym, że danego artysty nie ma w bazie.
13	Stworzenie raportu w aplikacji Flask dla artysty o ID 12345 (nieprawidłowy ID)	Zostały powtórzone kroki z poprzedniego testu, ze zmienionym wyłącznie ID artysty.	W przeglądarce pod adresem *localhost:5000/report* wyświetla się komunikat o tym, że podany ID jest błędny.

Rezultaty

1.

2.



3.

```
In [23]:
for artist in artists[:10]:
    albums_json = json.dumps(sp.artist_albums(str('spotify:artist:' + artist), limit = 50))

albums_json
```

Out[23]:

'{"href": "https://api.spotify.com/v1/artists/0Y5tJX1MQlPlqiwlOH1tJY/albums?offset=0&limit=50&include_groups=album,single,compilation,appears_on", "items": [{"album_group": "album", "album_type": "album", "artists": [{"external_urls": {"spotify": "https://open.spotify.com/artist/7A8S43ryYdbWpJKeHRZRcq"}, "href": "https://api.spotify.com/v1/artists/7A8S43ryYdbWpJKeHRZRcq", "id": "7A8S43ryYdbWpJKeHRZRcq", "name": "JACKBOYS", "type": "artist", "uri": "spotify:artist:7A8S43ryYdbWpJKeHRZRcq"}, {"external_urls": {"spotify": "https://open.spotify.com/artist/0Y5tJX1MQlPlqiwlOH1tJY"}, "href": "https://api.spotify.com/v1/artists/0Y5tJX1MQlPlqiwlOH1tJY", "id": "0Y5tJX1MQlPlqiwlOH1tJY", "name": "Travis Scott", "type": "artist", "uri": "spotify:artist:0Y5tJX1MQlPlqiwlOH1tJY"}], "available_markets": ["AD", "AE", "AG", "AL", "AM", "AO", "AR", "AT", "AU", "AZ", "BA", "BB", "BD", "BE", "BF", "BG", "BH", "BI", "BJ", "BN", "BO", "BR", "BS", "BT", "BW", "BY", "BZ", "CA", "CD", "CG", "CH", "CI", "CL", "CM", "CO", "CR", "CV", "CW", "CY", "CZ", "DE", "DJ", "DK", "DM", "DO", "DZ", "EC", "EE", "EG", "ES", "FI", "FJ", "FM", "FR", "GA", "GB", "GD", "GE", "GH", "GM", "GN", "GQ", "GR", "GT", "GW", "GY", "HK", "HN", "HR", "HT", "HU", "ID", "IE", "IL", "IN", "IQ", "IS", "IT", "JM", "JO", "JP", "KE", "KG", "KH", "KI", "KM", "KN", "KR", "KW", "KZ", "LA", "LB", "LC", "LI", "LK", "LR", "LS", "LT", "LU", "LV", "LY", "MA", "MC", "MD", "ME", "MG", "MH", "MK", "ML", "MN", "MO", "MR", "MT", "MU", "MV", "MW", "MX", "MY", "MZ", "NA", "NE", "NG", "NI", "NL", "NO", "NP", "NR", "NZ", "OM", "PA", "PE", "PG", "PH", "PK", "PL", "PS", "PT", "PW", "PY", "QA", "RO", "RS", "RU", "RW", "SA", "SB", "SC", "SE", "SG", "SI", "SK", "SL", "SM", "SN", "SR", "ST", "SV", "SZ", "TD", "TG", "TH", "TJ", "TL", "TN", "TO", "TR", "TT", "TV", "TW", "TZ", "UA", "UG", "US", "UY", "UZ", "VC", "VE", "VN", "VU", "WS", "XK", "ZA", "ZM", "ZW"], "external_urls": {"spotify": "https://open.spotify.com/album/1Sf8GsXG32t0jNrX11xqWx"}, "href": "https://api.spotify.com/v1/albums/1Sf8GsXG32t0jNrX11xqWx", "id": "1Sf8GsXG32t0jNrX11xqWx", "images": [{"height": 640, "url": "https://i.scdn.co/image/ab67616d0000b273dfc2f59568272de50a257f2f", "width": 640}, {"height": 300, "url": "https://i.scdn.co/image/ab67616d00001e02dfc2f59568272de50a257f2f", "width": 300}, {"height": 64, "url": "https://i.scdn.co/image/ab67616d00004851dfc2f59

4.

```
In [25]:
for artist in artists[:10]:
    tracks_json = json.dumps(sp.artist_top_tracks(str('spotify:artist:' + artist)))

tracks_json
```

Out[25]:

'{"tracks": [{"album": {"album_type": "album", "artists": [{"external_urls": {"spotify": "https://open.spotify.com/artist/3TVXtAsR1Inumwj472S9r4"}, "href": "https://api.spotify.com/v1/artists/3TVXtAsR1Inumwj472S9r4", "id": "3TVXtAsR1Inumwj472S9r4", "name": "Drake", "type": "artist", "uri": "spotify:artist:3TVXtAsR1Inumwj472S9r4"}], "external_urls": {"spotify": "https://open.spotify.com/album/3SpBlxme9WbeQdI9kx7KAV"}, "href": "https://api.spotify.com/v1/albums/3SpBlxme9WbeQdI9kx7KAV", "id": "3SpBlxme9WbeQdI9kx7KAV", "images": [{"height": 640, "url": "https://i.scdn.co/image/ab67616d0000b273cd945b4e3de57edd28481a3f", "width": 640}, {"height": 300, "url": "https://i.scdn.co/image/ab67616d00001e02cd945b4e3de57edd28481a3f", "width": 300}, {"height": 64, "url": "https://i.scdn.co/image/ab67616d00004851cd945b4e3de57edd28481a3f", "width": 64}], "name": "Certified Lover Boy", "release_date": "2021-09-03", "release_date_precision": "day", "total_tracks": 21, "type": "album", "uri": "spotify:album:3SpBlxme9WbeQdI9kx7KAV"}, "artists": [{"external_urls": {"spotify": "https://open.spotify.com/artist/3TVXtAsR1Inumwj472S9r4"}, "href": "https://api.spotify.com/v1/artists/3TVXtAsR1Inumwj472S9r4", "id": "3TVXtAsR1Inumwj472S9r4", "name": "Drake", "type": "artist", "uri": "spotify:artist:3TVXtAsR1Inumwj472S9r4"}, {"external_urls": {"spotify": "https://open.spotify.com/artist/0Y5tJX1MQlPlqiwlOH1tJY"}, "href": "https://api.spotify.com/v1/artists/0Y5tJX1MQlPlqiwlOH1tJY", "id": "0Y5tJX1MQlPlqiwlOH1tJY", "name": "Travis Scott", "type": "artist", "uri": "spotify:artist:0Y5tJX1MQlPlqiwlOH1tJY"}], "disc_number": 1, "duration_ms": 291175, "explicit": true, "external_ids": {"isrc": "USUG12104402"}, "external_urls": {"spotify": "https://open.spotify.com/track/40iJIUlhi6renaREYGeIDS"}, "href": "https://api.spotify.com/v1/tracks/40iJIUlhi6renaREYGeIDS", "id": "40iJIUlhi6renaREYGeIDS", "is_local": false, "is_playable": true, "name": "Fair Trade (with Travis Scott)", "popularity": 88, "preview_url": "https://p.scdn.co/mp3-preview/8b7b8af3300f40e0bdd75ed39de3011380799c5d?cid=774b29d4f13844c495f206cafdad9c86", "track_number": 6, "type": "track", "uri": "spotify:track:40iJIUlhi6renaREYGeIDS"}, {"album": {"album_type": "album", "artists": [{"external_urls": {"spotify": "https://open.spotify.com/artist/0Y5tJX1MQlPlqiwlOH1tJY"}, "href": "https://api.spotify.com/v1/artist

5.

```
In [40]: data.show()
```

```
[Stage 132:=====================================================> (196 + 4) / 200]

+------------------+--------------------+------------------+---------+----------------+--------------------+----------+--
----------------+-----------+----------+-----------+--------+----------+--------------------+----------+--
|           artist_id|            album_id|       artist_name|followers|artist_popularity|              genres|album_type|
album_name|release_date|total_tracks|duration_ms|explicit|track_id|track_name|track_popularity|
+------------------+--------------------+------------------+---------+----------------+--------------------+----------+--
----------------+-----------+----------+-----------+--------+----------+--------------------+----------+--
|00Z3UDoAQwzvGu13H...|6E8WvkEF5UzumaYEr...|       Skizzy Mars|   291381|              59|      indie pop rap|     album|I
Can't Take Me A...|  2018-11-29|           9|       null|    null|    null|      null|      null|
|00Z3UDoAQwzvGu13H...|6E8WvkEF5UzumaYEr...|       Skizzy Mars|   291381|              59|           pop rap|     album|I
Can't Take Me A...|  2018-11-29|           9|       null|    null|    null|      null|      null|
|00Z3UDoAQwzvGu13H...|6E8WvkEF5UzumaYEr...|       Skizzy Mars|   291381|              59|               rap|     album|I
Can't Take Me A...|  2018-11-29|           9|       null|    null|    null|      null|      null|
|00Z3UDoAQwzvGu13H...|6E8WvkEF5UzumaYEr...|       Skizzy Mars|   291381|              59|underground hip hop|     album|I
Can't Take Me A...|  2018-11-29|           9|       null|    null|    null|      null|      null|
|00sCATpEvwH48ays7...|0y66ZtW87HvOi6qPQ...|      Jonita Gandhi|   389362|              67|           desi pop|    single|Va
seegara And Zar...|  2020-02-20|           1|       null|    null|    null|      null|      null|
|00sCATpEvwH48ays7...|0y66ZtW87HvOi6qPQ...|      Jonita Gandhi|   389362|              67|   modern bollywood|    single|Va
seegara And Zar...|  2020-02-20|           1|       null|    null|    null|      null|      null|
```

6.

```
vagrant@node1:~$ hadoop fs -ls /user/wisniewskij/spotify/data
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.9.1-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 201 items
-rw-r--r--   1 vagrant supergroup          0 2022-01-03 10:24 /user/wisniewskij/spotify/data/_SUCCESS
-rw-r--r--   1 vagrant supergroup     398300 2022-01-03 10:23 /user/wisniewskij/spotify/data/part-00000-ee6a553b-f661-4179-a272-f57d6475b9f4-c000.csv
-rw-r--r--   1 vagrant supergroup     426375 2022-01-03 10:23 /user/wisniewskij/spotify/data/part-00001-ee6a553b-f661-4179-a272-f57d6475b9f4-c000.csv
-rw-r--r--   1 vagrant supergroup     432349 2022-01-03 10:23 /user/wisniewskij/spotify/data/part-00002-ee6a553b-f661-4179-a272-f57d6475b9f4-c000.csv
-rw-r--r--   1 vagrant supergroup     384109 2022-01-03 10:23 /user/wisniewskij/spotify/data/part-00003-ee6a553b-f661-4179-a272-f57d6475b9f4-c000.csv
vagrant@node1:~$ hadoop fs -cat /user/wisniewskij/spotify/data/part-00199-ee6a553b-f661-4179-a272-f57d6475b9f4-c000.csv
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.9.1-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
artist_id,artist_name,followers,artist_popularity,genres,album_id,album_type,album_name,release_date,total_tracks,track_id,track_name,explicit,track_popularity,duration_ms
007FXgr0jL8JxhPJjacXQH,Eric Land,469190,67,forro,4TBSaLZMxwDCPAt7QZ5xwv,single,Infelizes para Sempre (Ao Vivo),2020-12-18,1,"","","","",""
007FXgr0jL8JxhPJjacXQH,Eric Land,469190,67,sertanejo pop,4TBSaLZMxwDCPAt7QZ5xwv,single,Infelizes para Sempre (Ao Vivo),2020-12-18,1,"","","","",""
00FQb4jTyendYWaN8pK0wa,Lana Del Rey,14780949,87,art pop,7Cq3TWayv3b31UvZuzzICe,album,Lust For Life,2017-07-21,16,"","","","",""
00FQb4jTyendYWaN8pK0wa,Lana Del Rey,14780949,87,pop,7Cq3TWayv3b31UvZuzzICe,album,Lust For Life,2017-07-21,16,"","","","",""
00me4KelLsvMxtSkydlMyU,Cosculluela,4245307,73,latin,4PxaaXuXQk8Jamx5p8c102,single,HBD - Tu cumpleaños,2019-11-22,1,"","","","",""
00me4KelLsvMxtSkydlMyU,Cosculluela,4245307,73,latin hip hop,4PxaaXuXQk8Jamx5p8c102,single,HBD - Tu cumpleaños,2019-11-22,1,"","","","",""
00me4KelLsvMxtSkydlMyU,Cosculluela,4245307,73,perreo,4PxaaXuXQk8Jamx5p8c102,single,HBD - Tu cumpleaños,2019-11-22,1,"","","","",""
```

7.

```
In [23]: start = time.time()
         report1 = generate_genres('4O15NlyKLIASxsJ0PrXPfz')
         print("{:.2f}".format(time.time() - start) + 's')
         report1
```

```
his can cause serious performance degradation.
[Stage 61:================================================>       (177

14.03s
```

Out[23]:

	place	artist_name	followers	artist_popularity	common_genres
0	1	Drake	59456578	98	rap
1	2	Eminem	50298599	93	rap
2	3	Post Malone	35190445	92	melodic rap, rap
3	4	Juice WRLD	21177987	95	melodic rap
4	5	Travis Scott	19220791	93	rap
5	6	Kendrick Lamar	18889029	89	rap
6	7	Cardi B	18697678	86	rap

```
In [24]: start = time.time()
         report2 = generate_albums('4O15NlyKLIASxsJ0PrXPfz')
         print("{:.2f}".format(time.time() - start) + 's')
         report2
```

```
5.03s
```

Out[24]:

	album_name	release_date	total_tracks	album_type
0	Demon High	2021-10-29	1	single
1	V12 (feat. Lil Uzi Vert)	2021-09-14	1	single
2	Blue Notes 2 (feat. Lil Uzi Vert)	2021-09-01	1	single
3	Blue Notes 2 (feat. Lil Uzi Vert)	2021-08-31	1	single
4	Teenage Dream 2 (with Lil Uzi Vert)	2021-06-25	1	single
5	Sossboy 2 (feat. Lil Uzi Vert)	2021-06-09	1	single
6	Sossboy 2 (feat. Lil Uzi Vert)	2021-06-09	1	single
7	BADASS	2021-04-09	1	single
8	BADASS	2021-04-09	1	single

```
In [25]: start = time.time()
         report3 = generate_tracks('4O15NlyKLIASxsJ0PrXPfz')
         print("{:.2f}".format(time.time() - start) + 's')
         report3
```

```
3.91s
```

Out[25]:

	track_name	track_popularity	duration	explicit	album_name
0	XO Tour Llif3	86	03:02	True	Luv Is Rage 2
1	20 Min	86	03:40	True	Luv Is Rage 2 (Deluxe)
2	The Way Life Goes (feat. Oh Wonder)	83	03:41	True	Luv Is Rage 2
3	Drankin N Smokin	80	03:33	True	Pluto x Baby Pluto
4	Myron	79	03:44	True	Eternal Atake (Deluxe) - LUV vs. The World 2
5	Neon Guts (feat. Pharrell Williams)	79	04:18	True	Luv Is Rage 2
6	Money Longer	78	03:18	True	Lil Uzi Vert vs. The World
7	Erase Your Social	78	03:19	True	The Perfect LUV Tape
8	7AM	77	03:39	True	Luv Is Rage

8.

```
In [26]: start = time.time()
         report1 = generate_genres('4O15NlyKLIASxsJ0PrXPfz')
         print("{:.2f}".format(time.time() - start) + 's')
         report1
```

```
0.12s
```

Out[26]:

	place	artist_name	followers	artist_popularity	common_genres
0	1	Drake	59456578	98	rap
1	2	Eminem	50298599	93	rap
2	3	Post Malone	35190445	92	melodic rap, rap
3	4	Juice WRLD	21177987	95	melodic rap
4	5	Travis Scott	19220791	93	rap
5	6	Kendrick Lamar	18889029	89	rap
6	7	Cardi B	18697678	86	rap
7	8	Kanye West	15881092	94	rap
8	9	J. Cole	15262135	89	rap
9	10	Chris Brown	14980839	90	rap

```
In [27]: start = time.time()
         report2 = generate_albums('4O15NlyKLIASxsJ0PrXPfz')
         print("{:.2f}".format(time.time() - start) + 's')
         report2
```

```
0.13s
```

Out[27]:

	album_name	release_date	total_tracks	album_type
0	Demon High	2021-10-29	1	single
1	V12 (feat. Lil Uzi Vert)	2021-09-14	1	single
2	Blue Notes 2 (feat. Lil Uzi Vert)	2021-09-01	1	single
3	Blue Notes 2 (feat. Lil Uzi Vert)	2021-08-31	1	single
4	Teenage Dream 2 (with Lil Uzi Vert)	2021-06-25	1	single
5	Sossboy 2 (feat. Lil Uzi Vert)	2021-06-09	1	single
6	Sossboy 2 (feat. Lil Uzi Vert)	2021-06-09	1	single
7	BADASS	2021-04-09	1	single
8	BADASS	2021-04-09	1	single
9	Pluto x Baby Pluto	2020-11-13	16	album
10	Pluto x Baby Pluto (Deluxe)	2020-11-13	24	album

```
In [28]: start = time.time()
         report3 = generate_tracks('4O15NlyKLIASxsJ0PrXPfz')
         print("{:.2f}".format(time.time() - start) + 's')
         report3
```

```
0.07s
```

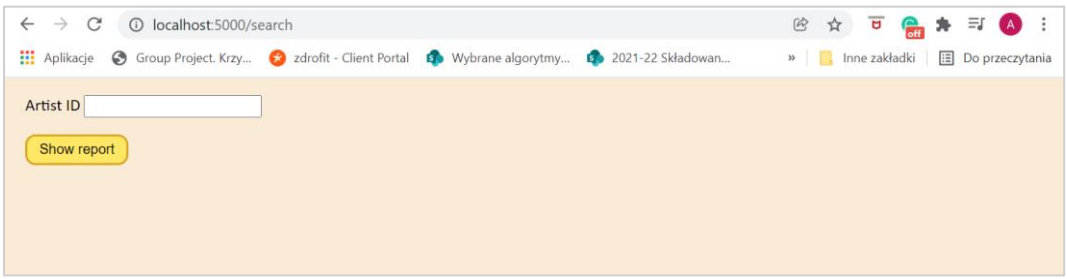
Out[28]:

	track_name	track_popularity	duration	explicit	album_name
0	XO Tour Llif3	86	03:02	True	Luv Is Rage 2
1	20 Min	86	03:40	True	Luv Is Rage 2 (Deluxe)
2	The Way Life Goes (feat. Oh Wonder)	83	03:41	True	Luv Is Rage 2
3	Drankin N Smokin	80	03:33	True	Pluto x Baby Pluto
4	Myron	79	03:44	True	Eternal Atake (Deluxe) - LUV vs. The World 2
5	Neon Guts (feat. Pharrell Williams)	79	04:18	True	Luv Is Rage 2
6	Money Longer	78	03:18	True	Lil Uzi Vert vs. The World
7	Erase Your Social	78	03:19	True	The Perfect LUV Tape
8	7AM	77	03:39	True	Luv Is Rage

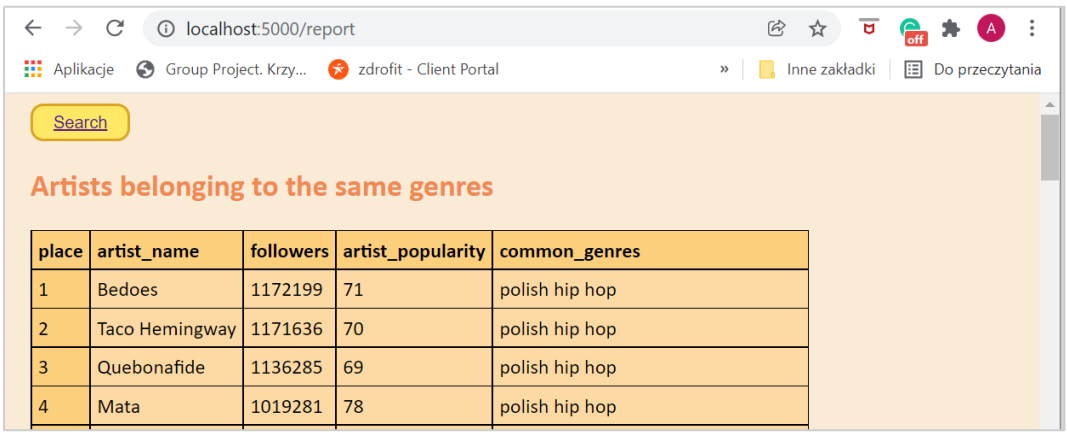
9.

```
In [29]: start = time.time()
         report1 = generate_genres('1SKeSGQ3LMHYCEgqFGvJbE')
         print("{:.2f}".format(time.time() - start) + 's')
         report1

         [Stage 78:==================================>                    (4 + 3) / 7]

         1.80s

Out[29]: -1

In [30]: start = time.time()
         report2 = generate_albums('1SKeSGQ3LMHYCEgqFGvJbE')
         print("{:.2f}".format(time.time() - start) + 's')
         report2

         [Stage 81:=========================================>             (5 + 2) / 7]

         1.45s

Out[30]: -1

In [31]: start = time.time()
         report3 = generate_tracks('1SKeSGQ3LMHYCEgqFGvJbE')
         print("{:.2f}".format(time.time() - start) + 's')
         report3

         [Stage 84:=========================================>             (5 + 2) / 7]

         1.49s

Out[31]: -1
```

10.

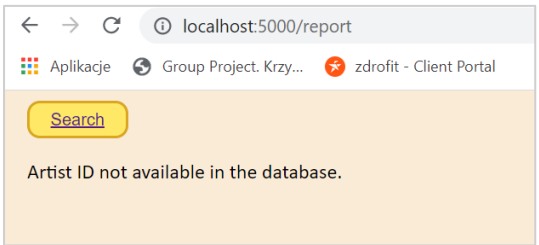


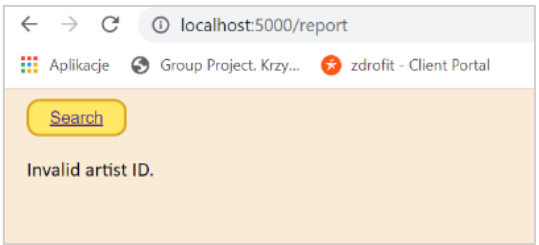
11.



place	artist_name	followers	artist_popularity	common_genres
1	Bedoes	1172199	71	polish hip hop
2	Taco Hemingway	1171636	70	polish hip hop
3	Quebonafide	1136285	69	polish hip hop
4	Mata	1019281	78	polish hip hop

12. & 13.





6 Podsumowanie

Podczas realizacji zadania, powstała aplikacja kliencka, która spełnia założenia postawione na początku projektu. Prosty i łatwy w obsłudze intefejs użytkownika umożliwia generowania widoków ze statystykami artystów zarejstrowanych w Spotify. Zostały obsłużone błędy, które mogą się zdarzyć, jeżeli użytkownik poda błędny identyfikator artysty. Dodatkowo, dzięki zastosowaniu Apache HBase, udało się znacznie zredukować czas wyświetlania raportu w aplikacji.

6.1 Podział pracy

Członek zespołu	Zakres pracy
Agata Makarewicz	❖ Pozyskiwanie danych (scraping) ❖ Przepływ danych w Apache NiFi ❖ Przetwarzanie danych źródłowych w Apache Spark ❖ Warstwa prezentacyjna (aplikacja Flask)
Jacek Wiśniewski	❖ Pozyskiwanie danych (spotipy) ❖ Przepływ danych w Apache NiFi ❖ Analiza danych i generowanie widoków wsadowych w Apache Spark ❖ Składowanie danych w Apache HBase