

POLITECHNIKA LUBELSKA
Wydział Elektrotechniki i Informatyki
Kierunek Informatyka



PROJEKT

Searchify

**Aplikacja do wyszukiwania i zapisywania danych utworów przy użyciu Spotify
API wykorzystująca technologię Docker**

Jakub Łabendowicz
Michał Łatwiński
Marcin Oskar Ludian

Lublin 2022

Wstęp	4
Narzędzia i technologie użyte do stworzenia aplikacji	4
Docker, Docker desktop i Docker Compose	4
PHP	4
SQL	4
REST API	4
Projekt systemu	4
Specyfikacja wymagań	4
Wymagania funkcjonalne	5
Strona wyszukiwarki	5
Strona rezultatów wyszukiwania	5
Strona ulubionych utworów	5
Wymagania нефункционалне	5
Dostępność/Niezawodność	5
Wydajność	5
Bezpieczeństwo	5
Wdrożenie	5
Użyteczność	5
Projekt interfejsów systemu	6
Implementacja	7
Docker desktop i Docker Compose	7
Serwis php	7
Serwis db	8
Serwis apache	8
Serwis phpmyadmin	8
PHP	9
SQL	9
Podsumowanie	10

Wstęp

Aplikacja umożliwia użytkownikowi wyszukiwanie utworów po wpisanej frazie, za pomocą integracji z Spotify Api. Usługa integracji wymaga autoryzacji użytkownika. Aplikacja wyświetla listę rezultatów wyszukiwania oraz zapewnia funkcjonalności jak odtworzenie wskazanego utworu, wyświetlenie strony utworu za pomocą klienta Spotify oraz zapis danych utworu do lokalnej bazy danych. Istnieje możliwość eksportu pobranych danych w formacie json oraz xml. Zapewniona została strona podglądu listy ulubionych utworów/utworów zapisanych w lokalnej bazie danych. Widnieją tam takie funkcje jak import danych do bazy z plików json lub xml oraz możliwość usunięcia utworu z ulubionych.

Narzędzia i technologie użyte do stworzenia aplikacji

Docker, Docker desktop i Docker Compose

otwarte oprogramowanie służące do realizacji wirtualizacji na poziomie systemu operacyjnego (tzw. „konteneryzacji”), działające jako „platforma dla programistów i administratorów do tworzenia, wdrażania i uruchamiania aplikacji rozproszonych¹.

Docker jest określany jako narzędzie, które pozwala umieścić program oraz jego zależności (biblioteki, pliki konfiguracyjne, lokalne bazy danych itp.) w lekkim, przenośnym, wirtualnym kontenerze, który można uruchomić na prawie każdym serwerze z systemem Linux. Kontenery wraz z zawartością działają niezależnie od siebie i nie wiedzą o swoim istnieniu. Mogą się jednak ze sobą komunikować w ramach ściśle zdefiniowanych kanałów wymiany informacji. Dzięki uruchamianiu na jednym wspólnym systemie operacyjnym, konteneryzacja jest lżejszym (mniej zasobochłonnym) sposobem wirtualizacji niż pełna wirtualizacja lub parawirtualizacja za pomocą wirtualnych systemów operacyjnych.

PHP

język programowania zaprojektowany do generowania stron internetowych i budowania aplikacji webowych w czasie rzeczywistym

SQL

strukturalny oraz deklaratywny język zapytań. Jest to język dziedziny używany do tworzenia, modyfikowania relacyjnych baz danych oraz do umieszczania i pobierania danych z tych baz.

REST API

rodzaj sieciowego interfejsu programowania aplikacji (ang. network-based API), w którym wykorzystuje się architekturę i protokoły sieci Web (w szczególności protokół HTTP) do komunikacji między aplikacjami znajdującymi się na oddzielnych urządzeniach w sieci.

Projekt systemu Specyfikacja wymagań

Wymagania funkcjonalne

Strona wyszukiwarki

- wyświetlanie formularza z polem wyszukiwanej frazy
- Wyświetlenie nawigacji do strony wyszukiwarki i ulubionych utworów

Strona rezultatów wyszukiwania

- wyświetlanie listy wyszukanych utworów
- wyświetlanie akcji odtworzenia, wyświetlenia i dodania do ulubionych obok każdego utworu
- Wyświetlenie nawigacji do strony wyszukiwarki i ulubionych utworów

Strona ulubionych utworów

- wyświetlanie listy ulubionych utworów
- wyświetlanie akcji odtworzenia, wyświetlenia i usunięcia z ulubionych obok każdego utworu
- Wyświetlenie nawigacji do strony wyszukiwarki i ulubionych utworów

Wymagania niefunkcjonalne

Dostępność/Niezawodność

- Aplikacja ma być dostępna w systemie 24/7/365

Wydajność

- Aplikacja musi obsługiwać jednocześnie 10 użytkowników

Bezpieczeństwo

- Użytkownik powinien mieć możliwość pobrania zgromadzonych danych i usunięcia ich z bazy danych

Wdrożenie

- Aplikacja powinna integrować się z publicznymi api umożliwiającymi dostęp do danych utworów
- Aplikacja powinna być dostępna lokalnie dla wskazanych użytkowników
- Aplikacja powinna być zaimplementowana w technologiach PHP i MySQL oraz Docker Compose

Użyteczność

- Aplikacja musi być posiadać przejrzysty design
- Aplikacja musi być responsywna
- Aplikacja ma działać zarówno na przeglądarce internetowej jak i urządzeniach mobilnych
- Aplikacja ma umożliwiać zmianę motywu strony

Projekt interfejsów systemu

Searchify!

Search

Favorites



Searchify!

Maryla Rodowicz



Rys. 1. Strona wyszukiwarki

Searchify!

Search

Favorites



Results



Niech Żyje Bał
Maryla Rodowicz



Małgośka
Maryla Rodowicz



Ale To Już Było
Andrzej Sikorowski



Wariatka Tańczy
Maryla Rodowicz

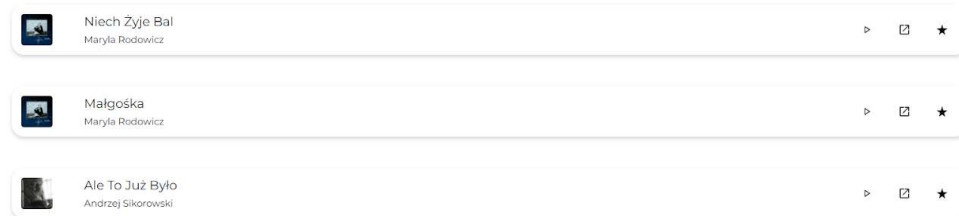


Sing-Sing
Maryla Rodowicz



Rys. 2. Strona rezultatów wyszukiwania

Favorites



Rys. 3. Strona ulubionych utworów

Implementacja

Docker desktop i Docker Compose

System wykorzystuje sieć backend oraz frontend, w których działają usługi php, db, apache oraz phpmyadmin.

Serwis php

```
php:
  build: './php-fpm/'
  container_name: 'php'
  networks:
    - backend
  depends_on:
    - db
  volumes:
    - ./public_html:/var/www/html/
```

Rys. 4. Fragment pliku docker-compose.yml pokazujący konfigurację usługi php

Serwis db

```
db:
  container_name: db
  image: mysql
  command: --default-authentication-plugin=mysql_native_password
  restart: always
  networks:
    - backend
  environment:
    MYSQL_ROOT_PASSWORD: pass
    MYSQL_DATABASE: searchify
    MYSQL_USER: user
    MYSQL_PASSWORD: pass
  volumes:
    - ./public_html/docs/sql_files/tracks.sql:/docker-entrypoint-initdb.d/tracks.sql
  ports:
    - "9906:3306"
```

Rys. 5. Fragment pliku docker-compose.yml pokazujący konfigurację usługi db

Serwis apache

```
apache:
  build: './apache/'
  depends_on:
    - php
    - db
  networks:
    - frontend
    - backend
  ports:
    - "8000:80"
  volumes:
    - ./public_html/:/var/www/html/
```

Rys. 6. Fragment pliku docker-compose.yml pokazujący konfigurację usługi apache

Serwis phpmyadmin

```
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  networks:
    - backend
  ports:
    - '8081:80'
  restart: always
  environment:
    PMA_HOST: db
  depends_on:
    - db
```

Rys. 7. Fragment pliku docker-compose.yml pokazujący konfigurację usługi phpmyadmin


```

1 FROM httpd
2 RUN apt-get update; \
3     apt-get upgrade;
4
5 COPY php.apache.conf /usr/local/apache2/conf/php.apache.conf
6 RUN echo "Include /usr/local/apache2/conf/php.apache.conf" >> /usr/local/apache2/conf/httpd.conf

```

Rys.8. Dockerfile dla konfiguracji httpd

```

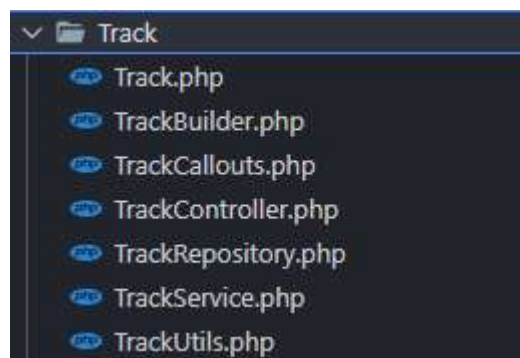
1 FROM php:8.1.1-fpm-alpine3.7
2 RUN apk update; \
3     apk upgrade;
4
5 RUN docker-php-ext-install mysqli

```

Rys. 9. Dockerfile dla konfiguracji php

PHP

Tworząc kod strony wykorzystaliśmy architekturę warstwową:

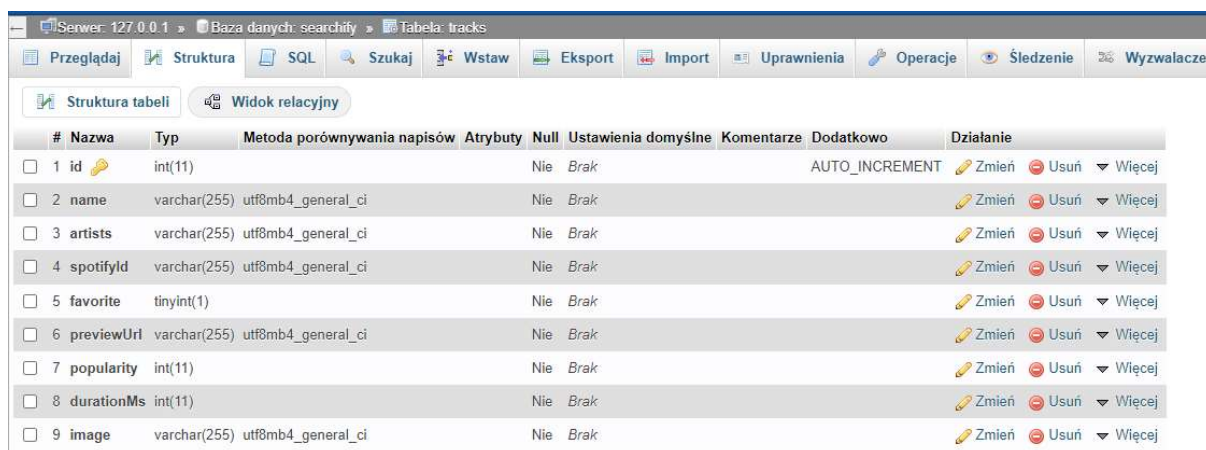


Rys. 8. Struktura plików aplikacji związana z modelem Track

Stworzyliśmy klasę modelu `Track.php` oraz `TrackBuilder.php`, która wykorzystując wzorzec projektowy Builder umożliwia tworzenie obiektu klasy `Track`. Z pomocą klasy `TrackRepository` wykonujemy wymagane zapytania do bazy danych, a klasa `TrackCallouts` umożliwia komunikację z api Spotify. Wyżej w hierarchii znajduje się klasa `TrackService`, która odpowiada za logikę biznesową aplikacji z wykorzystaniem metod klas `TrackRepository` oraz `TrackCallouts`. Klasa `TrackController` zapewnia metody potrzebne do wyświetlenia danych użytkownikowi, wykorzystuje ona metody klasy `TrackService` oraz klasy `TrackUtils`, która posiada metody użytkowe potrzebne do konwersji danych i ich przetwarzania.

SQL

Zaimplementowaliśmy jedną tabelę bazy danych pokazaną na obrazku poniżej:



	#	Nazwa	Typ	Metoda porównywania napisów	Atrybuty	Null	Ustawienia domyślne	Komentarze	Dodatkowo	Działanie
<input type="checkbox"/>	1	id	int(11)			Nie	Brak		AUTO_INCREMENT	Zmień Usuń Więcej
<input type="checkbox"/>	2	name	varchar(255)	utf8mb4_general_ci		Nie	Brak			Zmień Usuń Więcej
<input type="checkbox"/>	3	artists	varchar(255)	utf8mb4_general_ci		Nie	Brak			Zmień Usuń Więcej
<input type="checkbox"/>	4	spotifyId	varchar(255)	utf8mb4_general_ci		Nie	Brak			Zmień Usuń Więcej
<input type="checkbox"/>	5	favorite	tinyint(1)			Nie	Brak			Zmień Usuń Więcej
<input type="checkbox"/>	6	previewUrl	varchar(255)	utf8mb4_general_ci		Nie	Brak			Zmień Usuń Więcej
<input type="checkbox"/>	7	popularity	int(11)			Nie	Brak			Zmień Usuń Więcej
<input type="checkbox"/>	8	durationMs	int(11)			Nie	Brak			Zmień Usuń Więcej
<input type="checkbox"/>	9	image	varchar(255)	utf8mb4_general_ci		Nie	Brak			Zmień Usuń Więcej

Rys. 9. Struktura tabeli tracks w bazie danych

Podsumowanie

Udało się nam zrealizować wszystkie założenia projektowe. Aplikacja została stworzona przy użyciu narzędzia docker compose, które uruchomiło 4 usługi: php, apache, phpmyadmin oraz mysql. Interfejs graficzny aplikacji umożliwia łatwe wyszukiwanie utworów z platformy Spotify, a także dodawanie lub usuwanie ich z ulubionych. Serwer php komunikuje się z bazą wykorzystując mapowanie obiektowo-relacyjne, z tego powodu wykorzystaliśmy wzorzec projektowy builder. Dzięki komunikacji z bazą każdy użytkownik ma własną listę ulubionych utworów. Strona internetowa została zaprojektowana, tak aby była responsywna oraz zawierała prosty i przejrzysty design. Większość funkcjonalności jest realizowana przez przyciski, co pozwala zminimalizować błędy podczas użytkowania.