

2019/20 WIZ, Informatyka Zaawansowane technologie webowe - laboratorium			
Dokument końcowy projektu: Strona internetowa do grupowania i współdzielenia linków			
Arkadiusz Rasz 242493 Szałajko Karol 242557	Termin zajęć: Poniedziałek 15:15-16:45	Data oddania dokumentu: 15.06.2020r.	Ocena:

Spis Treści

Wstęp	2
Wprowadzenie do tematu	2
Zakres funkcji	2
Przypadki użycia	2
Omówienie funkcjonalności oraz zrzuty ekranów	4
Architektura aplikacji	11
Diagram pakietów	11
Diagram rozmieszczenia	12
Schemat bazy danych.....	14
Opis implementacji	15
Front-end	15
Back-end	16
Wykorzystane zewnętrzne API	17
Opis procesu implementacji	18
Kontrola wersji.....	18
Code Review.....	18
Continuous Integration oraz Continuous Delivery.....	18
Testy.....	20
Testy jednostkowe	20
Testy integracyjne.....	21
Podsumowanie	22

Wstęp

Wprowadzenie do tematu

Strona internetowa pozwala na tworzenie kolekcji odnośników do zewnętrznych stron. Kolekcje te mogą być prywatne, udostępniane dla innych użytkowników lub publiczne. Pozwala to na grupowanie materiałów zorientowanych tematycznie w jednym miejscu. Dla odnośników do wybranych stron internetowych dostępne są dodatkowe funkcjonalności.

Zakres funkcji

- dodawanie, usuwanie i modyfikowanie kolekcji
- dodawanie, usuwanie i modyfikowanie elementów kolekcji
- dodatkowe funkcjonalności dla wybranych linków
- udostępnianie kolekcji
- upublicznianie kolekcji
- zmiana motywu (ciemny / jasny)
- wybór ilości kolumn wyświetlania elementów
- sortowanie i wybieranie elementów
- wyświetlanie statystyk publicznych kolekcji
- panel administratora z możliwością zmiany ustawień i usuwaniem użytkowników

Przypadki użycia

Jako użytkownik chcę mieć możliwość tworzenia kolekcji, aby lepiej grupować odnośniki do stron.

Jako użytkownik chcę mieć możliwość usuwania moich kolekcji.

Jako użytkownik chcę mieć możliwość modyfikacji moich kolekcji.

Jako użytkownik chcę mieć możliwość zmiany ustawień prywatności kolekcji.

Jako użytkownik chcę mieć możliwość podglądu statystyk publicznej kolekcji.

Jako użytkownik chcę mieć możliwość udostępniania kolekcji dla innych użytkowników.

Jako użytkownik chcę mieć możliwość zmiany uprawnień użytkowników w udostępnionych przeze mnie kolekcjach.

Jako użytkownik chcę mieć możliwość usunięcia uprawnień użytkowników w udostępnionych przeze mnie kolekcjach.

Jako użytkownik chcę mieć możliwość dodawania elementów do kolekcji.

Jako użytkownik chcę mieć możliwość usuwania elementów z kolekcji.

Jako użytkownik chcę mieć możliwość modyfikacji elementów w kolekcji.

Jako użytkownik chcę mieć możliwość sortowania elementów w kolekcji na podstawie nazwy elementu, nazwy hosta, do którego odnosi się link.

Jako użytkownik chcę mieć możliwość wyboru z jakich stron będą wyświetlane elementy w kolekcji.

Jako użytkownik chcę mieć możliwość wyboru ilości kolumn, w których wyświetlane będą elementy kolekcji.

Jako użytkownik chcę mieć możliwość wyboru pomiędzy ciemnym i jasnym motywem aplikacji.

Jako użytkownik chcę mieć możliwość dodania utworu do kolejki na platformie Spotify, jeśli link wskazuje na piosenkę na tej platformie.

Jako użytkownik chcę mieć możliwość odtworzenia wideo, jeśli link wskazuje na film z platformy YouTube.

Jako użytkownik chcę mieć możliwość podglądu zdjęcia, jeśli link wskazuje na plik zdjęciowy.

Jako użytkownik chcę mieć możliwość zalogowania się na swoje konto.

Jako niezarejestrowany użytkownik chcę mieć możliwość rejestracji w systemie.

Jako administrator chcę mieć możliwość modyfikacji maksymalnej liczby kolekcji, którą może posiadać użytkownik.

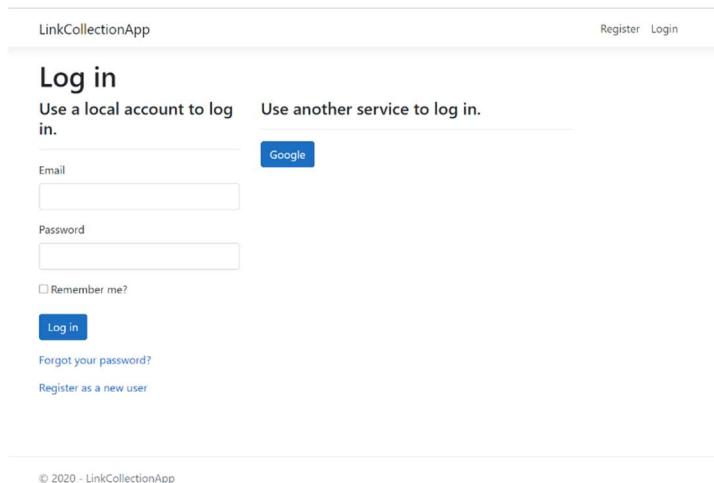
Jako administrator chcę mieć możliwość modyfikacji maksymalnej liczby elementów w kolekcji.

Jako administrator chcę mieć możliwość tymczasowego usuwania użytkowników z serwisu.

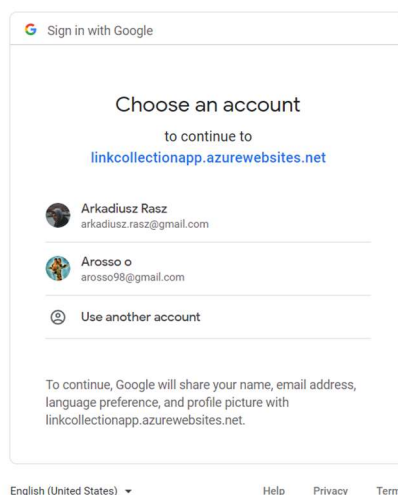
Omówienie funkcjonalności oraz zrzuty ekranów

Autoryzacja

Autoryzacja użytkownika została zaimplementowana z użyciem technologii ASP.Net Core Identity, która posiada domyślne wbudowane ekrany i logikę odpowiadającą procesem rejestracji oraz logowania, jak i zarządzaniem kontem użytkownika. Dostępna jest opcja logowania przez serwis Google, po wciśnięciu którego następuje przekierowanie do zewnętrznego serwisu.



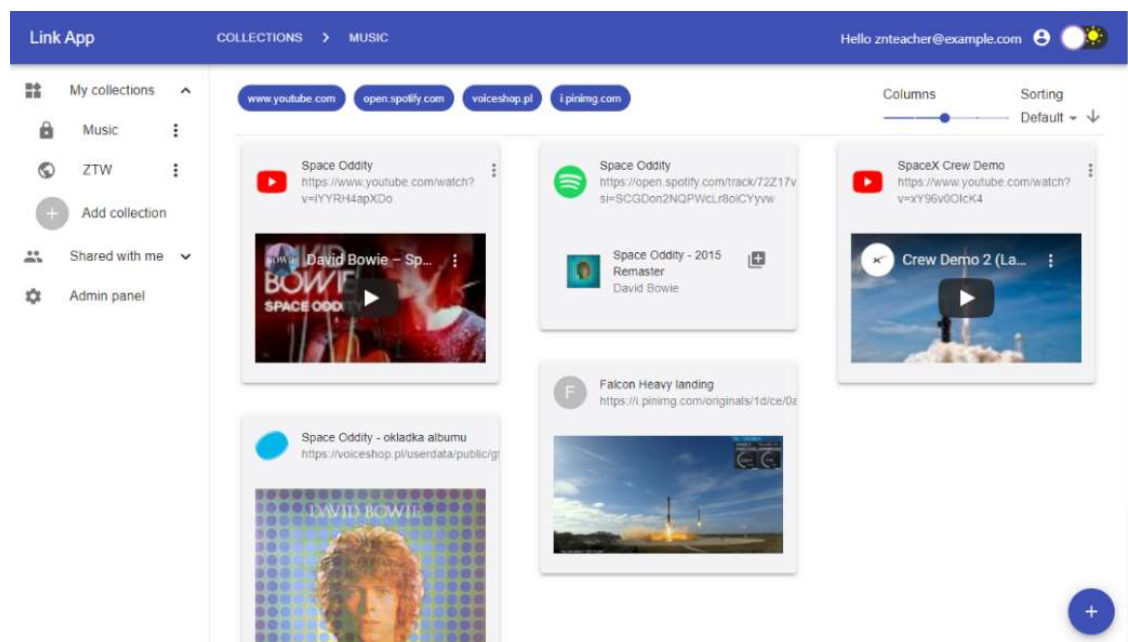
Rysunek 1 - Ekran logowania do aplikacji



Rysunek 2 - Autoryzacja przez serwis Google

Kolekcje odnośników

Po zalogowaniu, użytkownik ma dostęp do utworzonych przez niego kolekcji, ma możliwość ich edycji oraz tworzenia nowych. Elementy kolekcji zawierają informacje o nazwie elementu, ikony oraz adresu url do strony zewnętrznej. Dla wybranych linków wyświetlane są dodatkowe informacje oraz zapewnione są dodatkowe funkcjonalności.

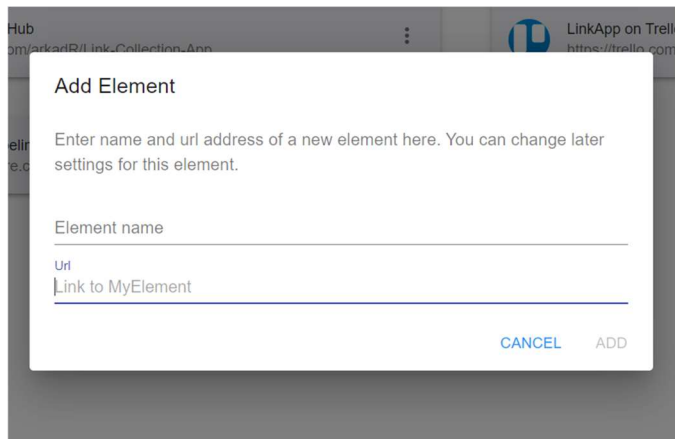


Rysunek 3 - Widok kolekcji

Zarządzanie elementami

Dla dodawania, usuwania oraz modyfikacji kolekcji oraz elementów wykorzystywane są okna dialogowe z wbudowaną walidacją wejścia użytkownika. Dialogi te są wyświetlane po wybraniu odpowiedniej opcji w menu kontekstowych lub po wybraniu przycisków dodawania w postaci FAB. W przypadku niepoprawnych lub braku danych, przycisk zatwierdzenia jest zablokowany.

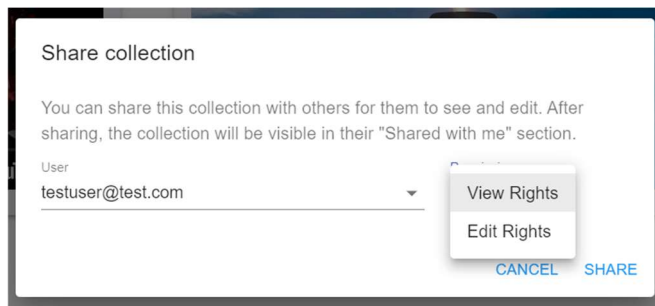
Walidacja jest powtórzona również po stronie serwera. Endpointy zabezpieczone są również przed żądaniami o modyfikację elementów innych użytkowników.



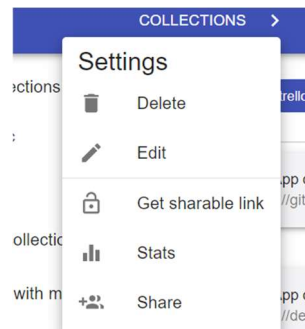
Rysunek 4 - Dialog dodawania elementu

Udostępnianie kolekcji

Dostępna jest funkcjonalność udostępniania kolekcji innym użytkownikom. Użytkownik, któremu kolekcja została udostępniona, w zależności od posiadanych uprawnień, może wyświetlać lub edytować kolekcję.

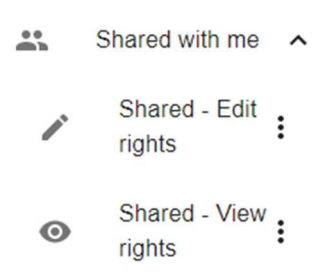


Rysunek 5 - Widok udostępniania kolekcji



Rysunek 6 - Przykład menu kontekstowego

Lista udostępnionych użytkownikowi kolekcji widoczna jest w osobnej sekcji w bocznym menu. Do zaznaczenia, jakie uprawnienia posiada użytkownik, użyte są odpowiednie ikony.

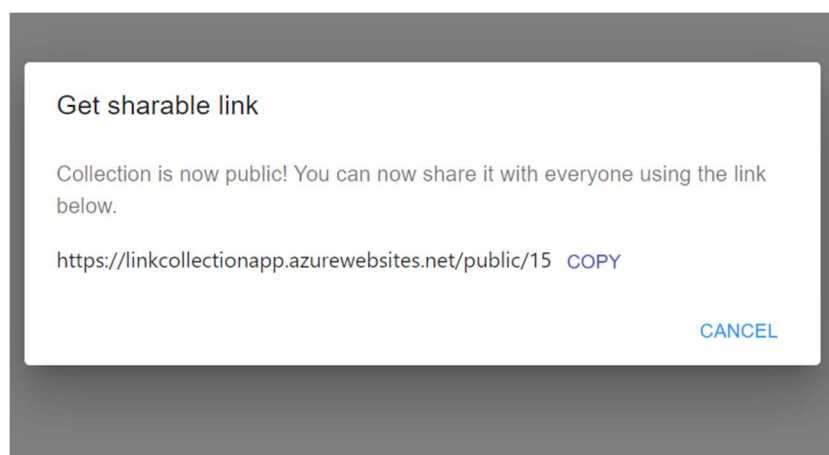


Rysunek 7 - Lista udostępnionych kolekcji

Publiczne kolekcje

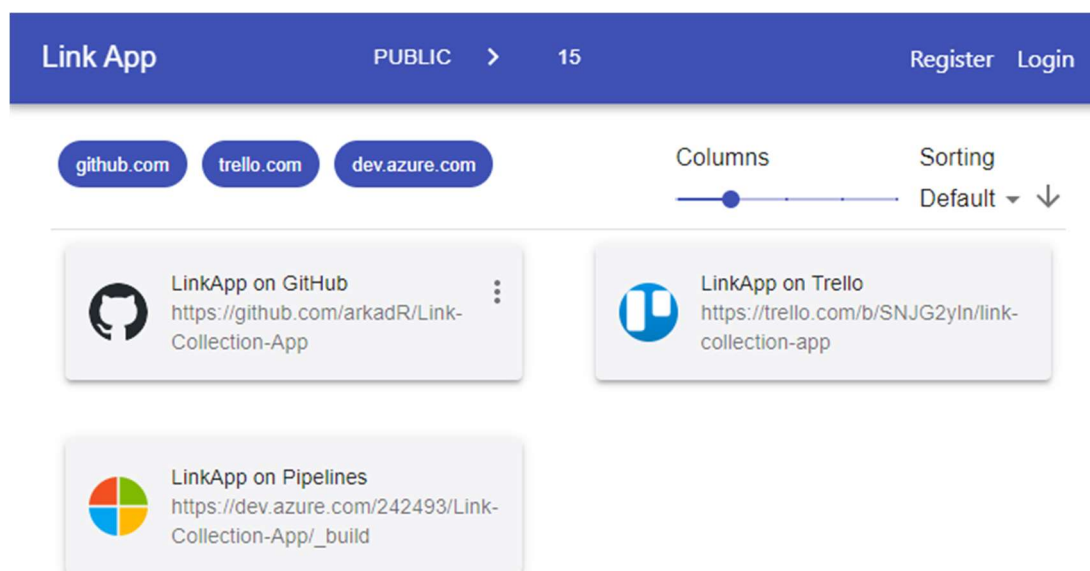
Kolekcja może zostać oznaczona przez właściciela jako publiczna, dzięki czemu każda osoba posiadająca link do tej kolekcji będzie w stanie ją wyświetlić bez konieczności logowania się do serwisu.

Po wybraniu opcji upublicznienia kolekcji, wyświetla się okno dialogowe z możliwością skopiowania linku do schowka, aby łatwo go udostępnić.



Rysunek 8 - Widok upubliczniania kolekcji

Widok dla niezalogowanego użytkownika po przejściu do kolekcji za pomocą udostępnionego linku jest identyczny jak w przypadku zalogowanego użytkownika, lecz nie ma możliwości dodawania ani modyfikacji elementów, a pasek boczny z listą kolekcji nie jest dostępny.

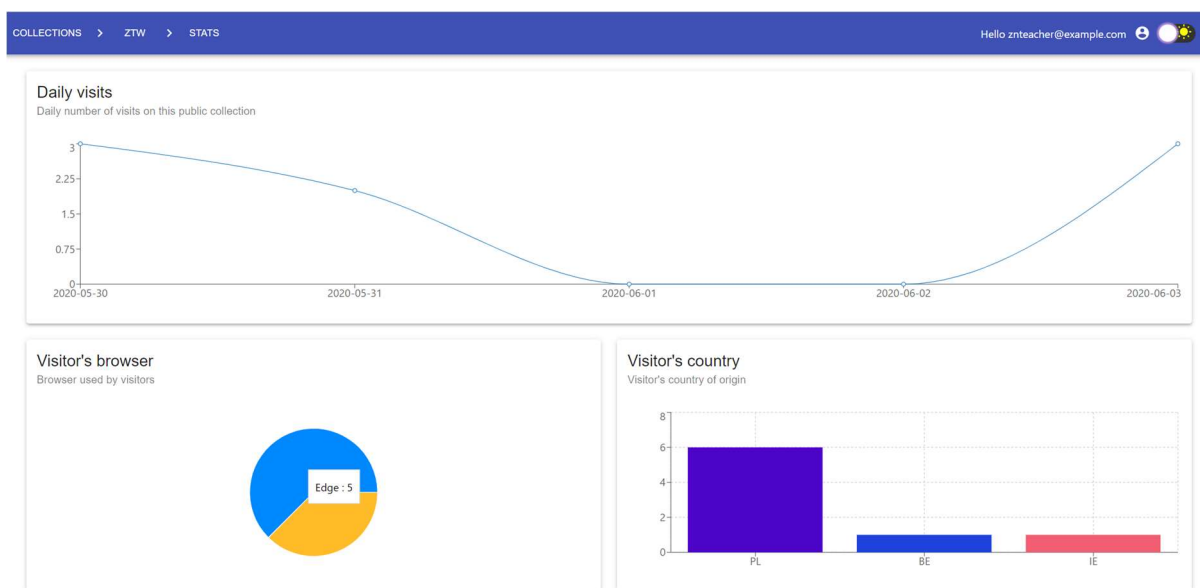


Rysunek 9 - Widok publicznej kolekcji dla niezalogowanego użytkownika

Publiczne kolekcje zawierają dodatkowy panel statystyk w postaci wykresów, przedstawiających liczbę wejść w zależności od czasu, przeglądarki z jakich korzystali odwiedzający oraz kraje pochodzenia odwiedzających.

Dane wydobywane są z informacji sekcji header zapytania użytkowników, jak i po ich adresie IP z użyciem zewnętrznego serwisu IpInfo.

Dane potrzebne do wyświetlania wykresów mogą być kosztowne do policzenia, dlatego są one przechowywane w pamięci podręcznej przez jeden dzień. Pozwala to na zmniejszenie liczby zapytań do bazy danych jak i zmniejszenie czasu oczekiwania na wynik zapytania kosztem braku możliwości częstego odświeżania wykresów.



Rysunek 10 - Panel ze statystykami wejść w publiczną kolekcję

Panel Administracyjny

Użytkownik z uprawnieniami administratora posiada również opcję skorzystania z panelu administratora. Dostępne są tutaj opcje dotyczące konfiguracji aplikacji dla wszystkich użytkowników – maksymalna liczba kolekcji na użytkownika oraz liczba elementów w kolekcji.

Tabela umieszczona niżej zawiera listę zarejestrowanych użytkowników wraz z ich adresami email oraz id w bazie danych. Administrator ma możliwość zablokowania użytkowników z korzystania z aplikacji na 30 dni. Po zablokowaniu, zostają oni wylogowani bez możliwości ponownej autoryzacji.

Link App

ADMIN

Hello znteacher@example.com

Number of collections

Maximum number of collections a user can have is set to 100

CHANGE

Number of elements

Maximum number of elements in the collection is set to 100

CHANGE

Users

Current number of users is 6

Name	Email	Id	Actions
testuser@gmail.com	testuser@gmail.com	692c9949-b1a2-41ef-8278-2dab43c15acc	
arkadiusz.rasz@gmail.com	arkadiusz.rasz@gmail.com	7ffc27de-9e11-4e9b-b625-12c302d26508	🔒
testuser@test.com	testuser@test.com	d0127d5c-46c5-4dac-8a13-c7752c77a160	
znteacher@example.com	znteacher@example.com	d2e7428b-1378-4d46-995f-877ed92caa3b	🔒
242557@student.pwr.edu.pl	242557@student.pwr.edu.pl	fac3df1c-f3cc-4b56-a8ba-7354b4ec2189	🔒

Rysunek 11 - Widok panelu administratora

LinkCollectionApp

Register Login

Locked out

This account has been locked out, please try again later.

Rysunek 12 - Próba logowania przez zablokowanego użytkownika

Dodatkowe funkcjonalności wybranych elementów

Youtube

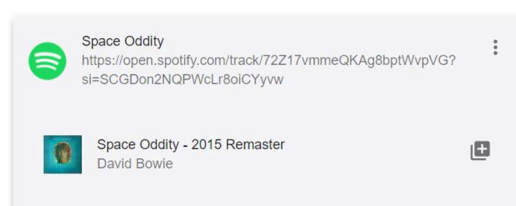
Dla odnośników do strony internetowej youtube.com dostępny jest osadzony odtwarzacz wideo, pozwalający na odtwarzanie załączonego filmu bez konieczności przechodzenia do zewnętrznej strony.



Rysunek 13 - Element z odnośnikiem do filmu na stronie youtube.com

Spotify

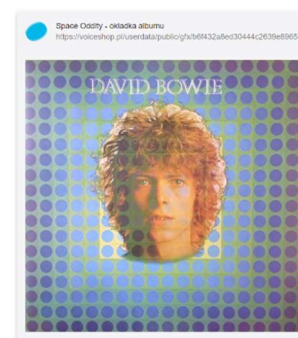
Dla odnośników do utworów na platformie spotify.com, możliwe jest dodanie tego utworu do kolejki użytkownika. Dodatkowo wyświetlana jest okładka albumu oraz nazwa utworu i wykonawca.



Rysunek 14 - Element z odnośnikiem do utworu na stronie spotify.com

Zdjęcia

Dla odnośników które wskazują na zdjęcia (formaty jpeg, jpg, gif lub png) w elemencie wyświetlane jest podane zdjęcie.

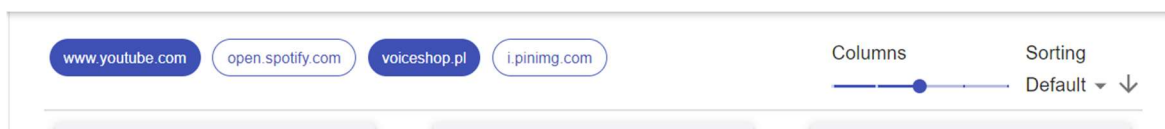


Rysunek 15 - Element z odnośnikiem do zdjęcia

Panel ustawień

Elementy w kolekcji można sortować wg domyślnej kolejności, po nazwie elementu oraz po nazwie hosta, do którego odnosi się link. Sortownie może być rosnące lub malejące. Istnieje możliwość wygaszenia odpowiednich elementów na podstawie nazwy hosta, do którego odnosi się link. Zablockowane hosty wyświetlane są na białym tle. Funkcjonalność ta pozwala szybko znaleźć elementy w dużych kolekcjach.

Po wybraniu jednego z hostów, wszystkie odpowiadające elementy są chowane.



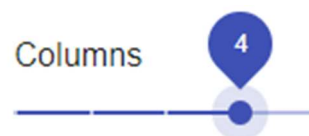
Rysunek 16 - Panel do sortowania oraz wybierania hostów elementów

Preferencje użytkownika

Poniższe preferencje użytkownika zapisywane są w ciasteczkach.

Liczba kolumn w kolekcji

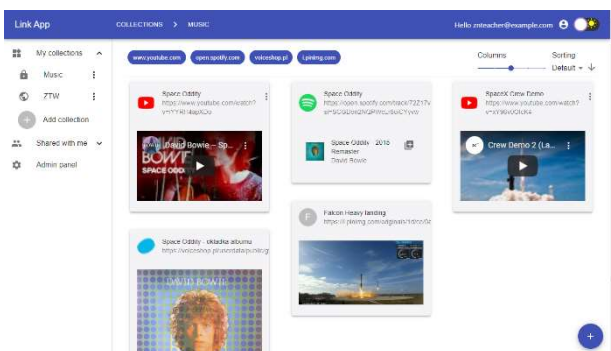
Użytkownik ma możliwość wyboru liczby kolumn w których będą wyświetlane elementy w kolekcji (1-5).



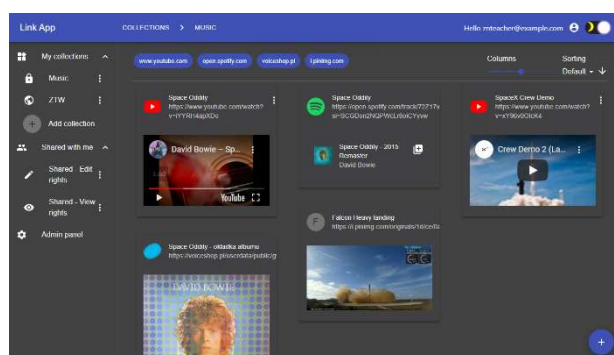
Rysunek 17 - suwak zmiany liczby kolumn w kolekcji

Wybór motywu

Za pomocą przełącznika użytkownik ma możliwość wyboru motywu aplikacji – ciemnego bądź jasnego.



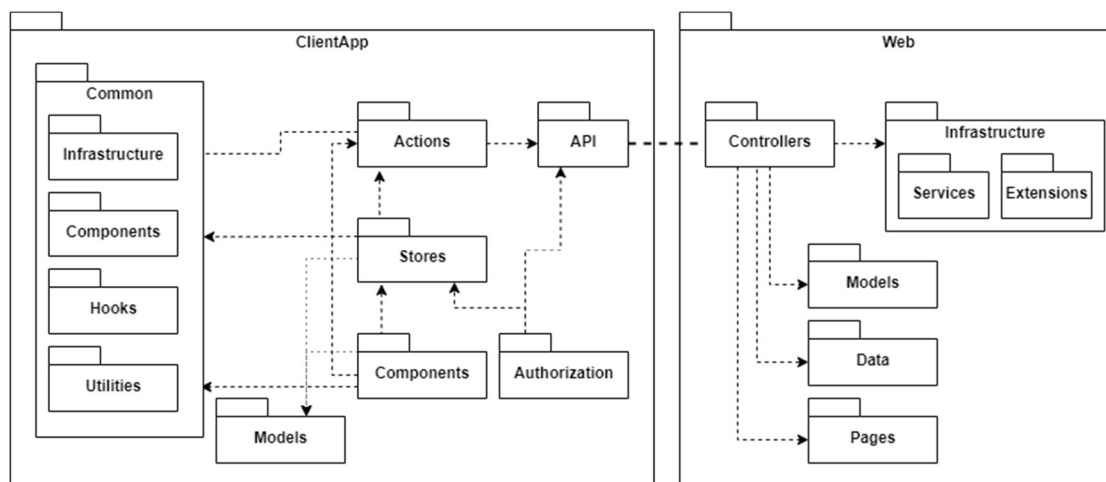
Rysunek 19 - aplikacja z jasnym motywem



Rysunek 18 - Aplikacja z ciemnym motywem

Architektura aplikacji

Diagram pakietów



Rysunek 20 - Diagram pakietów

Aplikacja składa się z dwóch głównych komponentów, odpowiadających za odpowiednio aplikację kliencką oraz serwis webowy w postaci API, który udostępnia funkcjonalności.

Aplikacja kliencka

Część interfejsowa napisana jest w technologii React z użyciem najpopularniejszego wzorca architektonicznego tej technologii – Flux. Elementy tego wzorca zawarte są w pakietach Actions oraz Stores. Actions posiada funkcje, które wywoływane są po wykonanej akcji użytkownika i wykonują odpowiednie działania. Wyniki działań dostarczane są do odpowiednich Storów, które jednocześnie przechowują stan danych. Po zmianie stanu, komponenty z niego korzystające są automatycznie aktualizowane nowym stanem.

Pliki służące do komunikacji z serwisem webowym zawarte są z pakiecie API. Są one głównie wykorzystywane przez akcje, które zwykle mają zmodyfikować lub zczytać bazę danych. Zawarte tutaj funkcje udostępniają również metody potrzebne na autoryzację oraz autentykację użytkownika.

Pakiety Components oraz Authorization zawierają komponenty biblioteki React, będące odpowiedzialne za interfejs użytkownika. Components zawiera widoki główne aplikacji, a Authorization widoki związane z użytkownikiem i stanem logowania.

Modele są kopią tych, znajdujących się na serwerze. Korzysta z nich większość warstw aplikacji. Same w sobie nie udostępniają żadnej funkcjonalności, służą jedynie jako nośniki danych dla innych komponentów.

Pakiet Common zawiera wszystkie pliki, które wykorzystywane są przez większą liczbę komponentów. Zawarte są tutaj również funkcje wymagane przez wzorec Flux, jak i dodatkowe implementacje komponentów typu hooks.

Serwis webowy

Głównymi elementami są kontrolery, które służą do komunikacji aplikacji z serwisem. Nie okazała się tutaj potrzebna dodatkowa warstwa dostępu do danych, jej rolę pełni Entity Framework.

Wyodrębniona logika jak i integracja z usługami zewnętrznymi znajdują się w pakiecie Services. Są one dostarczane do kontrolerów, które je wymagają, za pomocą dependency injection.

Pakiety Models oraz Data zawierają cały model domenowy. Models przechowuje modele wykorzystywane przez kontrolery, które są również definicją danych w bazie danych. Mapowanie danych z użyciem Entity Framework zdefiniowane jest w pakiecie Data.

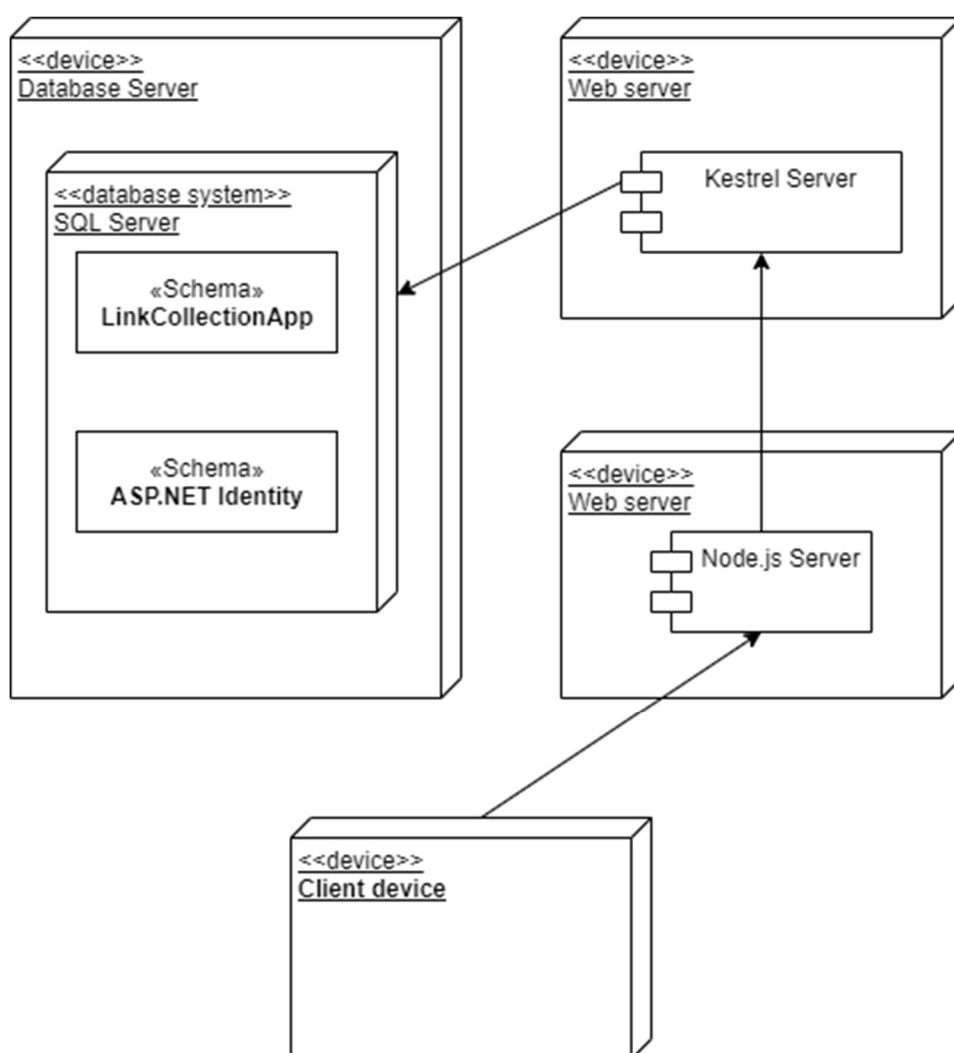
Diagram rozmieszczenia

Środowisko implementacyjne

Architektura podzielona jest na 3 moduły:

- Serwer bazy danych – Przechowuje dane o kolekcjach oraz autoryzacji użytkowników
- Kestrel Server – Uruchomiony serwer ASP.NET Core, opisany wyżej
- Node.js Server – Odpowiedzialny za dostarczanie interfejsu użytkownikowi oraz reagowanie na jego akcje. Wywołuje logikę biznesową na serwerze Kestrel

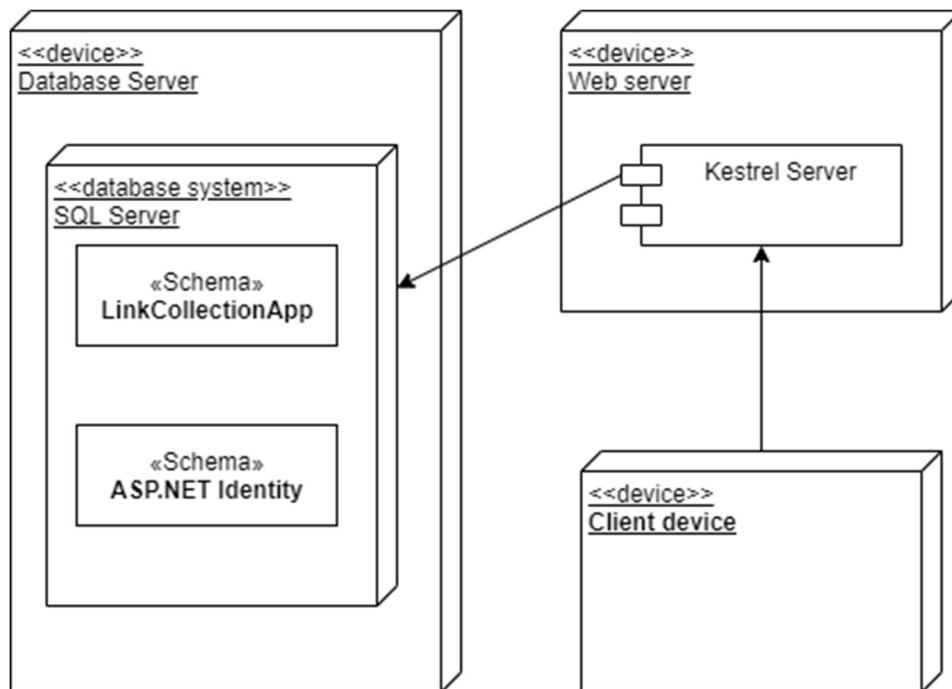
Wykorzystanie tutaj dodatkowego serwisu w postaci serwera Node.js pozwala na szybkie zmiany w interfejsie użytkownika bez konieczności przebudowania i zrestartowania całej aplikacji.



Rysunek 21 - Diagram rozmieszczenia w środowisku implementacyjnym

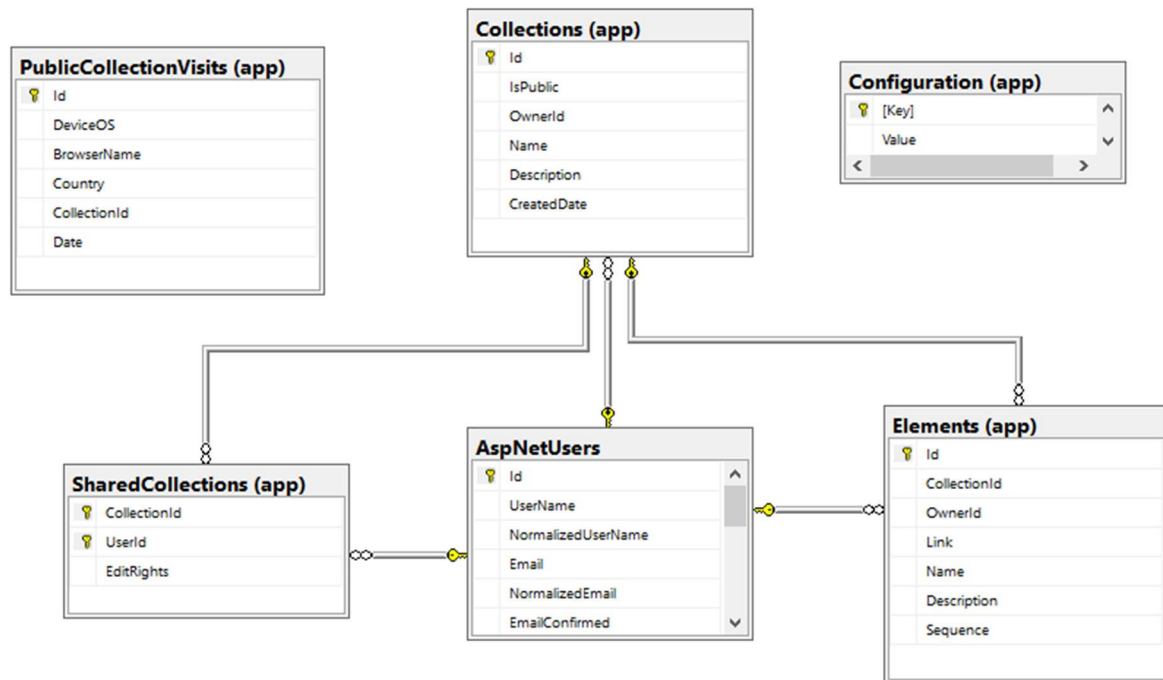
Środowisko wdrożeniowe

Różnicą na środowisku wdrożeniowym jest brak serwisu Node.js. Przed wdrożeniem, pliki aplikacji klienckiej są kompilowane i pakowane do bibliotek składających się z plików javascript oraz html. Plik ten jest dostarczany bezpośrednio z serwera Kestrel.



Rysunek 22 - Diagram rozmieszczenia w środowisku wdrożeniowym

Schemat bazy danych



Rysunek 23 - Schemat bazy danych

Głównymi tabelami wykorzystywanymi w aplikacji są **AspNetUsers**, **Collections**, **SharedCollections** oraz **Elements**. Jest to główna domena aplikacji, na której wykonywane są modyfikacje w ramach głównych funkcjonalności aplikacji.

PublicCollectionVisits zawiera surowe dane o wejściach użytkowników w kolekcje publiczne. Służą one jedynie do analizy danych w postaci wyświetlania ich na wykresach oraz nie są one częścią domeny.

Tabela **Configuration** służy jako słownik do przechowywania konfiguracji aplikacji, jak wartości ustawiane przez administratora.

Poza wymienionymi tabelami, baza danych zawiera również tabele wykorzystywane przez Identity w celach autoryzacji oraz autentykacji. Tabele te połączone są z **AspNetUsers**. Aby uprościć diagram, nie zostały one przedstawione.

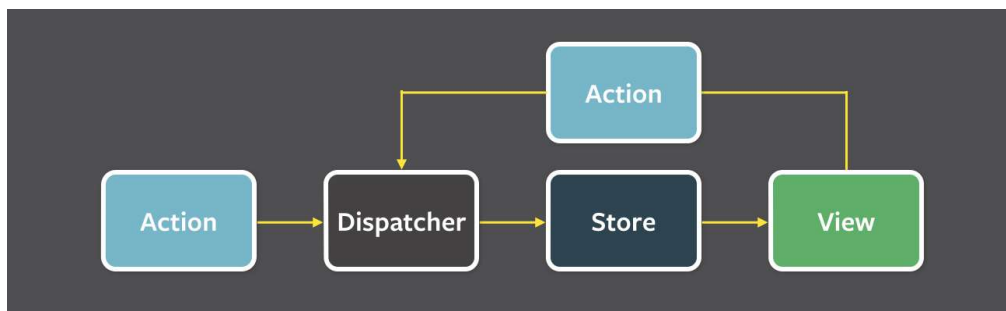
Opis implementacji

Front-end

Wzorzec architektoniczny Flux

Flux architektura aplikacji, która wykorzystuje jednokierunkowy przepływ danych. Zawiera następujące elementy:

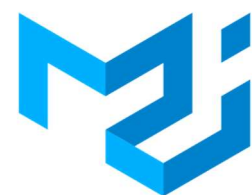
- Actions – Metody służące do przekazywania danych i informacji o zmianie stanów do dispatchera
- Dispatcher – Przyjmuje akcje i rozprowadza dane do zarejestrowanych callbacków w postaci storów
- Stores – Przechowują stan i logikę aplikacji, otrzymują informacje za pomocą callbacków z dispatcherów
- Components – Komponenty biblioteki React, które pobierają stan ze storów i przekazują je do elementów widocznych na ekranie.



Rysunek 24 - Przepływ danych we wzorcu Flux

Biblioteka Material-Ui dostarczająca elementy interfejsu

Biblioteka Material-Ui została wybrana do tworzenia elementów graficznych interfejsu. Biblioteka ta jest wspierana i rozwijana przez firmę Google. Zawiera bardzo dużo powszechnie używanych, konfigurowalnych komponentów. Biblioteka ta oparta jest na zasadach spisanych w MaterialDesign.



Back-end

Autentykacja oraz Autoryzacja

Proces autentykacji oraz autoryzacji zarządzany jest przez technologię Identity Server. Jest ona dobrze zintegrowana z ASP.Net Core, przez co dodanie tej funkcjonalności nie wymagało ręcznego tworzenia dodatkowych interfejsów oraz endpointów do zarządzania procesem logowania czy rejestracji.

Identity Server dodawany jest poprzez konfigurowanie odpowiednich serwisów oraz wykorzystaniem dostępnego szkieletu dla tabel w bazie danych.

Do autoryzacji wykorzystywane są tokeny JWT, które od długiego czasu są standardem w technologiach webowych.

Dostępna jest również funkcjonalność autoryzacji za pomocą konta Google.

Mapowanie obiektowo-relacyjne

Do mapowania danych przechowywanych w bazie danych na obiekty wykorzystywane podczas działania aplikacji wykorzystany został Entity Framework Core. Posłużył on również do generowania skryptów narostowych dla zmian w bazie danych podczas implementacji aplikacji.

Dependency Injection

Część serwerowa aplikacji używa wbudowanego w ASP.Net Core Dependency Injection tam, gdzie tylko to możliwe. Wszystkie serwisy dostarczające funkcjonalność oraz logikę wstrzykiwane są do kontrolerów dla każdego utworzonego kontekstu. Pozwala to na czystą architekturę, możliwość dalszego rozwoju aplikacji oraz umożliwienie poprawnych testów jednostkowych oraz integracyjnych z użyciem atrap obiektów.

Wszystkie dodatkowe serwisy oraz zastosowane technologie korzystają z dependency injection poprzez skonfigurowanie ich w pliku startowym.

Wykorzystane zewnętrzne API

Autoryzacja Google

Poza wbudowaną autoryzację poprzez ASP.Net Core Identity, aplikacja umożliwia również logowanie się za pomocą konta Google. Wykorzystywana jest biblioteka `Microsoft.AspNetCore.Authentication.Google`. Id aplikacji oraz client secret w środowisku deweloperskim przechowywane są w Visual Studio Safe Storage, aby nie były widoczne w kodzie jak i w kontroli wersji. W środowisku wdrożeniowym na Azure przechowywane są jako bezpieczne zmienne środowiskowe, skąd są widoczne jedynie dla aplikacji.

IpInfo

Usługa IpInfo pozwala na zebranie informacji o adresie IP, takich jak region i kraj pochodzenia czy dostawca Internetu. Usługa bez licencji pozwala na tysiące zapytań dziennie, co jest nieosiągalne przez naszą aplikację, a więc brak licencji nie jest problemem. Nie jest dostępna żadna biblioteka dla języka C#, dlatego wykonywane są zapytania http do publicznego API serwisu.

Usługa ta wykorzystywana jest dla pobierania danych o wejściach w kolekcje publiczne, które następnie wyświetlane są autorowi w postaci wykresów.

Spotify

Linki odnoszące się do odpowiednich utworów w serwisie Spotify zostały wyposażone w możliwość dodania ich do kolejki. Autoryzacja użytkownika opiera się na protokole OAuth 2.0. Użytkownik zostaje przekierowany na stronę Spotify, gdzie ma możliwość przyznania uprawnień naszej aplikacji. Następnie serwis Spotify przekierowuje na wskazany w zapytaniu link do naszego serwisu, przekazując niezbędny token. Po otrzymaniu aktywnego tokena użytkownik ma możliwość dodawania utworów do swojej kolejki oraz wyświetlana jest okładka albumu wraz z nazwą utworu oraz wykonawcą.

Darmowa licencja nie pozwala na używanie api serwisu Spotify w celach komercyjnych.

Youtube

Utworzone odnośniki do serwisu youtube.com wyświetlają się wraz z odtwarzaczem YouTube, co pozwala na odtwarzanie wideo bezpośrednio w aplikacji bez potrzeby przekierowania w dany link. Funkcjonalność ta została zaimplementowana z użyciem publicznego API udostępnianego przez Google.

FaviconGrabber

Usługa FaviconGrabber pozwala na zebranie informacji o ikonach favicon obecnych na danej stronie internetowej. W odpowiedzi otrzymywany jest plik JSON wraz z linkami do dostępnych ikon danej strony. Za pomocą uzyskanych linków do ikon wyświetlane są one przy każdym elemencie.

Opis procesu implementacji

Kontrola wersji

Wybrany systemem kontroli wersji na czas implementacji aplikacji jest git, a repozytorium hostowane jest publicznie na github.com pod adresem

<https://github.com/arkadR/Link-Collection-App>.

Wysyłanie zmian na gałąź główną kontroli wersji w używanym repozytorium jest zablokowana. Aby zmiany zostały zatwierdzone i trafiły na gałąź główną, musi zostać utworzony Pull Request z innej gałęzi, na której zostały zaimplementowane zmiany z daną funkcjonalnością. Zatwierdzenie Pull Requesta jest możliwe tylko w przypadku, kiedy jedna osoba zrobi przegląd kodu oraz zmiany nie spowodują błędów na środowisku CI (opisane niżej).

Skomplikowany proces dodawania zmian pozwala na utrzymanie lepszej jakości kodu oraz uniknięcia dalszych błędów

Code Review

Jak wspomniane wyżej, każde wprowadzenie zmiany wymaga utworzenia Pull Requesta, którego zmiany muszą być zatwierdzone przez drugą osobę z zespołu. W przypadku znalezionych błędów, wymagane są zmiany zanim funkcjonalność zostanie zaakceptowana.

Wykonywanie Code Review ułatwia zaznajomienie się z kodem wykonanego przez kolaborantów, wyłapanie błędów, jak i umożliwienie sugestii poprawy czy innego sposobu implementacji. Wszystko to pozwala na utrzymanie jakości kodu, szybsze odnajdywanie błędów, jak i również uczenia się na podstawie kodu drugiej osoby.

Continuous Integration oraz Continuous Delivery

Do ciągłej integracji poprzez budowę projektu oraz przeprowadzania testów użyty został serwis Azure Pipelines. Proces uruchamia się po każdej zmianie w systemie kontroli wersji, jak i po utworzeniu pull requesta. W przypadku niepoprawnego zakończenia procesu dla pull requesta, zatwierdzenie go nie jest możliwe.

Proces różni się w zależności od gałęzi, na którą wprowadzane są zmiany.

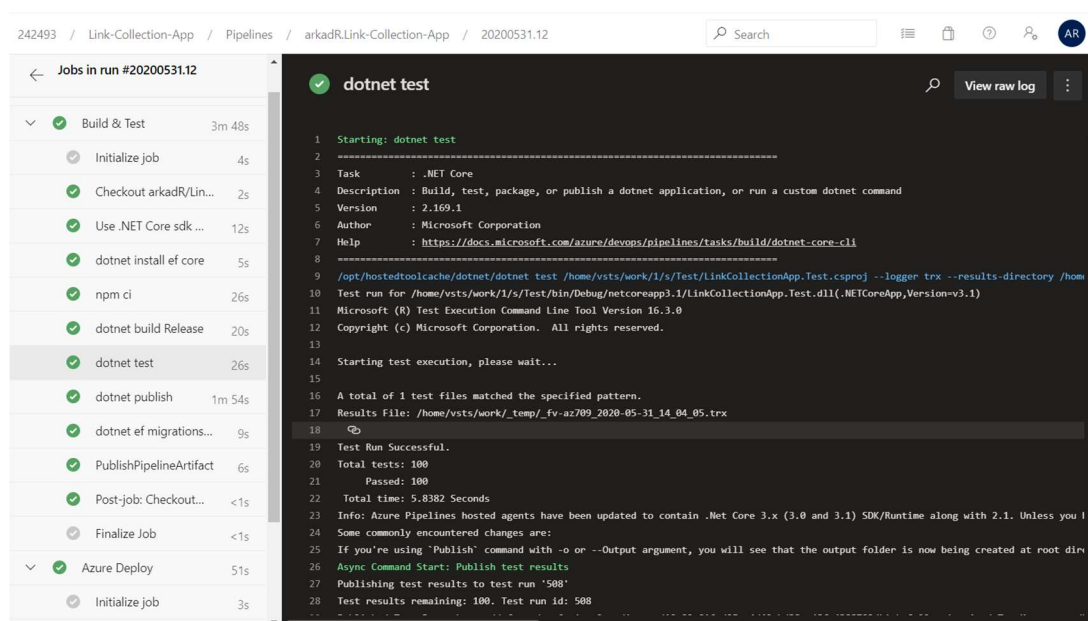
Kroki procesu dla gałęzi pobocznych:

1. **Checkout arkadR/Link-Collection-App@branch** – pobiera wersję projektu z nowymi zmianami
2. **Use .NET Core sdk 3.1.100** – wymusza używanie wersji .Net Core 3.1 w dalszych krokach
3. **dotnet install ef core** – instaluje globalnie narzędzia ef core na czas budowy projektu. Będzie ono wymagane przez utworzenie skryptów narostowych
4. **npm ci** – pobiera użyte paczki npm użyte przez aplikację po stronie front-end, jak i odnajduje błędy i zagrożenia w kodzie
5. **dotnet build Release** – buduje projekt w wersji Release
6. **dotnet test** – wykorzystuje pliki z wcześniej zbudowanego projektu oraz wykonuje na nich automatyczne testy jednostkowe oraz integracyjne
7. **dotnet publish** – opakuje projekt React w paczkę składającą się z plików Javascript oraz HTML umożliwiającą uruchomienie w przeglądarce bez dodatkowych narzędzi

Dla gałęzi master **dodatkowo**:

8. **dotnet ef migrations script** – przekształca pliki migracji utworzone przez ef core na skrypty narostowe w języku SQL
9. **PublishPipelineArtifact** – udostępnia zbudowane pliki oraz skrypty do dalszych kroków wykonywanych na innym środowisku
10. **DownloadArtifacts** – pobiera pliku udostępnione w poprzednim kroku
11. **Publish to Azure** – wdraża zbudowany projekt na platformę Azure, aktualizując wersję aplikacji
12. **Update Azure database** – wykonuje utworzone skrypty narostowe na produkcyjnej bazie danych, zachowując istniejące już dane

Dodatkowe kroki 10 oraz 11 udostępniające oraz pobierające pliki wymagane są przez użycie dwóch maszyn wirtualnych w procesie. Kroki wdrażania aplikacji na Azure dostępne są jedynie na platformie Windows. Maszyna wirtualna z systemem Linux jednak lepiej radzi sobie z budową projektu oraz wykonywaniem testów, zmniejszając czas wykonywania o prawie połowę. Z tego powodu zdecydowano się na użycie dwóch osobnych maszyn wirtualnych w tych krokach.



The screenshot displays the Azure Pipelines interface. On the left, a list of jobs is shown for the pipeline 'arkadR/Link-Collection-App'. The 'dotnet test' job is highlighted. On the right, the detailed view of the 'dotnet test' job is shown, indicating it was successful. The log content includes the following information:

```
1 Starting: dotnet test
2 -----
3 Task       : .NET Core
4 Description : Build, test, package, or publish a dotnet application, or run a custom dotnet command
5 Version    : 2.169.1
6 Author     : Microsoft Corporation
7 Help       : https://docs.microsoft.com/azure/devops/pipelines/tasks/build/dotnet-core-cli
8 -----
9 /opt/hostedtoolcache/dotnet/dotnet test /home/vsts/work/1/s/Test/LinkCollectionApp.Test.csproj --logger trx --results-directory /hom
10 Test run for /home/vsts/work/1/s/Test/bin/Debug/netcoreapp3.1/LinkCollectionApp.Test.dll(.NETCoreApp,Version=v3.1)
11 Microsoft (R) Test Execution Command Line Tool Version 16.3.0
12 Copyright (c) Microsoft Corporation. All rights reserved.
13
14 Starting test execution, please wait...
15
16 A total of 1 test files matched the specified pattern.
17 Results File: /home/vsts/work/_temp/_fv-az709-2020-05-31_14_04_05.trx
18
19 Test Run Successful.
20 Total tests: 100
21 Passed: 100
22 Total time: 5.8382 Seconds
23 Info: Azure Pipelines hosted agents have been updated to contain .Net Core 3.x (3.0 and 3.1) SDK/Runtime along with 2.1. Unless you l
24 Some commonly encountered changes are:
25 If you're using 'Publish' command with -o or --Output argument, you will see that the output folder is now being created at root dir
26 Async Command Start: Publish test results
27 Publishing test results to test run '508'
28 Test results remaining: 100. Test run id: 508
```

Rysunek 25 - Widok wykonanych kroków na platformie Azure Pipelines

Testy

Testy jednostkowe

Testy zostały napisane z użyciem biblioteki do testowania XUnit. Zaletą biblioteki nad wbudowanymi narzędziami testowania w środowisku Visual Studio jest możliwość wykonywania testów z parametrami. Pozwala to na napisanie jednego testu, który następnie sprawdzony będzie dla różnych przypadków brzegowych bez duplikacji kodu. W sumie w aplikacji znajduje się 100 przypadków testowych.

Do tworzenia atrap obiektów zastosowana została biblioteka Moq. Baza danych jest imitowana przez implementację bazy danych w pamięci, która tworzona jest i wypełniana danymi indywidualnie dla każdego wykonywanego testu.

Testy jednostkowe zostały napisane na wykorzystywane serwisy oraz kontrolery udostępniające funkcjonalność dla aplikacji po stronie front-end.

Akcje wykonywane na testowanych kontrolerach wykonywane są w osobnych transakcjach.

Przykład testu jednostkowego wraz z utworzeniem mocku wykorzystywanego przez testowany system obiektu:

```
[Fact]
public void UpdateElement_CollectionNotOwnedByUserButHasViewRights_Forbidden()
{
    //Arrange
    var userId1 = NewGuid();
    var userId2 = NewGuid();
    var newLink = NewGuid();
    var updateData = new ElementUpdateData { Link = newLink };
    AddUser(userId1);
    AddUser(userId2);
    var collection = AddCollection(userId1, 50);
    ShareCollection(collection.Id, userId2, false);
    Element element = null;
    InTransaction(context =>
    {
        element = context.Element.First();
    });

    //Act
    IActionResult result = null;
    InTransaction(context =>
    {
        var controller = new ElementsController(
            context,
            GetUserProviderMock(userId2),
            getCollectionConfigurationProviderMock());
        result = controller.UpdateElement(collection.Id, element.Id, updateData);
    });

    //Assert
    result.Should().BeOfType<ForbidResult>();
}
```

Rysunek 26 - Przykładowy test jednostkowy

Przykład utworzenia atrapy obiektu:

```
protected IUserContextProvider GetUserProviderMock(string userId)
{
    var userProviderMock = new Mock<IUserContextProvider>();
    userProviderMock.Setup(m => m.GetCurrentUserId()).Returns(userId);
    return userProviderMock.Object;
}
```

Rysunek 27 - Przykład tworzenia atrapy dla obiektów w testach

Testy integracyjne

Zaimplementowane testy integracyjne sprawdzają, czy endpointy zwracają poprawne kody http w zależności od statusu autoryzacji użytkownika. W szczególności, sprawdzane są endpointy publiczne, które powinny zwracać kod OK nawet dla niezalogowanego użytkownika oraz endpointy zabezpieczone, które powinny zwracać kod 401 lub 403 w przypadku braku autoryzacji.

Testowany serwer tworzony jest jako obiekt TestServer, który posiada odpowiednią konfigurację WebHostBuildera, symulującą środowisko wdrożeniowe. Jak w przypadku testów jednostkowych, baza danych jest imitowana przez jej implementację w pamięci.

Przykładowy test dla endpointów zabezpieczonych:

```
[Theory]
[InlineData("/api/collections")]
[InlineData("/api/sharedCollections")]
[InlineData("/api/sharedCollections/contributors")]
[InlineData("/api/configuration")]
[InlineData("/api/users")]
[InlineData("/api/users/me")]
public async Task Get_EndpointsSecuredWithAuthorization_Unauthorized(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.StatusCode.Should().Be(HttpStatusCode.Unauthorized);
}
```

Rysunek 28 - Przykład testu integracyjnego

Podsumowanie

Uważamy, że projekt został zakończony sukcesem, prace toczyły się zgodnie z harmonogramem zaproponowanym na początkowych zajęciach. Równolegle z implementacją aplikacji trwały prace nad testami jednostkowymi oraz integracyjnymi, jak i automatyzacją integracji oraz wdrażania za pomocą procesów CI oraz CD. Ustalony przez nas proces implementacji pozwolił na utrzymanie jakości kodu na dobrym poziomie, jak i lepszą komunikację i ustalenia nad sposobem implementacji. Aplikacja jest wdrożona na platformę Azure, a zaktualizowanie jej do nowszej wersji jest zautomatyzowane.

Aplikację można znaleźć pod linkiem <https://linkcollectionapp.azurewebsites.net/>

Zaplanowana na zajęcia aplikacja została skończona, lecz jest możliwe dalsze jej rozwijanie oraz usprawnianie interfejsu, na co nie starczyło czasu na laboratoriach. Zastosowana infrastruktura pozwala na łatwe rozwijanie aplikacji.