Implementacja systemu i wykonanie testów sprawdzających

Etap IV | Grupa nr.3

|  |  |
| --- | --- |
| **Produkt etapu** | **Autor** |
| Implemetacja bazy danych | Tomasz Jarnutowski |
| Implementacja warstwy logicznej | Daniel Gromak |
| Implementacja GUI | Daniel Gromak |
| Gotowy system informatyczny | Daniel Gromak |
| Wyniki przeprowadzonych testów | Daniel Gromak |

Spis treści

[1. Implementacja bazy danych 4](#_Toc125652997)

[1.1. Diagram bazy danych 4](#_Toc125652998)

[1.2. Sposób implementacji bazy danych 5](#_Toc125652999)

[Tabela User 6](#_Toc125653000)

[Tabela Photo 6](#_Toc125653001)

[Tabela Album 7](#_Toc125653002)

[Tabela AlbumPhoto 7](#_Toc125653003)

[Tabela Tag 8](#_Toc125653004)

[Tabela TagPhoto 8](#_Toc125653005)

[Tabela Favourite 8](#_Toc125653006)

[2. Implementacja warstwy logicznej 9](#_Toc125653007)

[2.1. Filtry routingu 9](#_Toc125653008)

[PrivateRoute 9](#_Toc125653009)

[Kod PrivateRoute 9](#_Toc125653010)

[OnlyAnonymousRoute 9](#_Toc125653011)

[Kod OnlyAnonymousRoute 10](#_Toc125653012)

[2.2. Logowanie 10](#_Toc125653013)

[Formularz logowania 10](#_Toc125653014)

[Funkcja loginUser 11](#_Toc125653015)

[Logowanie po stronie backendu 11](#_Toc125653016)

[2.3. Rejestracja 12](#_Toc125653017)

[Formularz rejestracji 12](#_Toc125653018)

[Funkcja submitHandler 12](#_Toc125653019)

[Rejestracja po stronie back-endu 13](#_Toc125653020)

[2.4. Dodawanie zdjęcia 13](#_Toc125653021)

[Formularz dodawania zdjęcia 14](#_Toc125653022)

[Funkcja addPhoto 15](#_Toc125653023)

[Dodawanie zdjęcia po stronie backendu 15](#_Toc125653024)

[2.5. Edycja zdjęcia 16](#_Toc125653025)

[Formularz edycji zdjęcia 17](#_Toc125653026)

[Funkcja editPhoto 17](#_Toc125653027)

[Edycja zdjęcia po stronie back-endu 18](#_Toc125653028)

[2.6. Dodawanie albumu 18](#_Toc125653029)

[Formularz dodawania albumu 18](#_Toc125653030)

[Funkcja addAlbum 19](#_Toc125653031)

[Dodawanie albumu po stronie back-endu 19](#_Toc125653032)

[2.7. Edycja albumu 20](#_Toc125653033)

[Formularz edycji albumu 21](#_Toc125653034)

[Funkcja editAlbum 21](#_Toc125653035)

[Edycja albumu po stronie back-endu 22](#_Toc125653036)

[2.8. Dodawanie tagów 23](#_Toc125653037)

[Dodawanie tagu/tagów po stronie backendu 23](#_Toc125653038)

[2.9. Dodawanie zdjęcia jako ulubione 23](#_Toc125653039)

[Oznaczanie zdjęcia jako ulubione z poziomu front-endu 23](#_Toc125653040)

[3. Implementacja GUI 24](#_Toc125653041)

[3.1. Strona logowania 24](#_Toc125653042)

[3.2. Strona rejestracji 25](#_Toc125653043)

[3.3. Strona główna 26](#_Toc125653044)

[3.4. Zdjęcia 26](#_Toc125653045)

[3.4.1. Podgląd zdjęcia 27](#_Toc125653046)

[3.4.2 Dodawanie zdjęcia 28](#_Toc125653047)

[3.4.3. Edycja zdjęcia 29](#_Toc125653048)

[3.4.4. Usuwanie zdjęcia 29](#_Toc125653049)

[3.5. Albumy 31](#_Toc125653050)

[3.5.1. Podgląd albumu 31](#_Toc125653051)

[3.4.2. Dodawanie albumu 31](#_Toc125653052)

[3.4.3. Edycja albumu 32](#_Toc125653053)

[3.4.4. Usuwanie albumu 32](#_Toc125653054)

[3.6. Galeria / profil użytkownika 32](#_Toc125653055)

[4. Gotowy system informatyczny 33](#_Toc125653056)

[5. Wyniki przeprowadzonych testów 33](#_Toc125653057)

[Testy funkcjonalności 34](#_Toc125653058)

[Zwrócenie przez API kodu 200 oraz listy zdjęć. 34](#_Toc125653059)

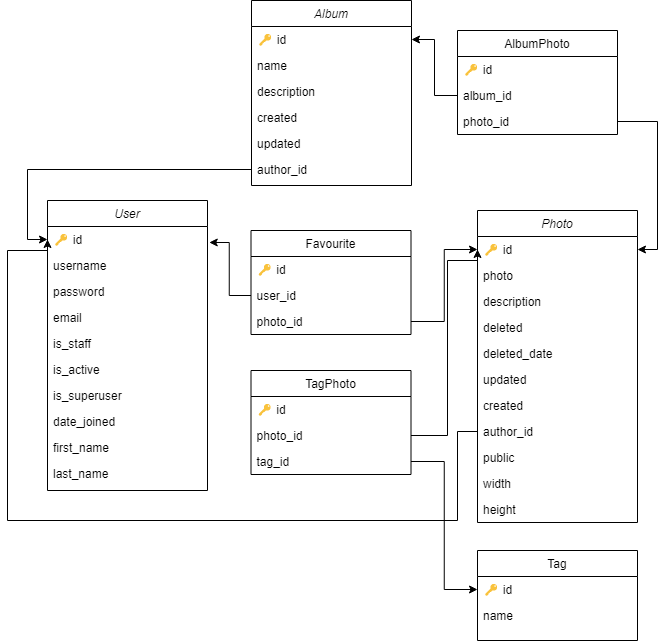
[Test krokowy: dostęp do ulubionych zdjęć po zalogowaniu. 34](#_Toc125653060)

[Otrzymanie danych o zdjęciu. 35](#_Toc125653061)

[Testy obciążeniowe 35](#_Toc125653062)

# Implementacja bazy danych

## Diagram bazy danych



## Sposób implementacji bazy danych



Baza została zaimplementowana z użyciem DjangoORM. Tabele są reprezentowana za pomocą klas, a zmienne odpowiadają nazw kolumn. Funkcje „\_\_str\_\_” pozwalają na łatwiejszy odczyt wpisów w przypadku użycia domyślnego panelu administracyjnego. DjangoORM domyślnie sam dodaje kolumnę „ID”, dlatego nie ich w implementacji powyżej. Na powyższym podglądzie nie ma również tabeli User, jest to spowodowane faktem że w projekcie jest użyty domyślny model użytkownika który jest dostarczany przez framework.

Tabele określające relacje wiele do wielu są tworzone z poziomu zmiennej klasy(models.ManyToManyField) i nie ma potrzeby tworzenia dodatkowej klasy. Relacja typu jeden do wielu jest reprezentowaną za pomocą models.ForeignKey. W przypadku pól typu dat, używam opcji „auto\_now” – która aktualizację datę przy każdej edycji wpisu oraz „auto\_now\_add” która dodaję datę w momencie tworzenia wpisu.

### Tabela User

Tabela zawierająca dane o użytkowniku. Jest to domyślna tabela dla użytkownika w frameworku Django

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| id | Klucz główny, integer | Identyfikator |
| username | varchar, unikalne | Nazwa użytkownika |
| password | varchar | Hasło |
| email | varchar, unikalne | Email |
| is\_staff | bool | Czy jest moderatorem |
| is\_active | bool | Czy konto jest aktywne |
| is\_superuser | boll | Czy jest administratorem |
| date\_joined | datetime | Data utworzenia konta |
| first\_name | varchar | Imię |
| last\_name | varchar | Nazwisko |

### Tabela Photo

Tabela zawierająca informacje o zdjęciu. Posiada klucz obcy author\_id odnoszący się do tabeli User (Relacja typu OneToMany)

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| id | Klucz główny, integer | Identyfikator |
| photo | varchar | Ścieżka do pliku |
| description | text | Opis |
| public | boolean | Czy zdjęcie jest publiczne |
| deleted | boolean | Czy plik znajduje sie w koszu |
| deleted\_date | datetime | Data umieszczenia pliku w koszu |
| updated | datetime | Data ostatniej aktualizacji |
| created | datetime | Data utworzenia |
| author\_id | Klucz obcy, integer | Id autora zdjęcia |
| width | integer | Wysokość zdjęcia |
| height | integer | Szerokość zdjęcia |

### Tabela Album

Tabela zawierająca informacje o albumie. Posiada klucz obcy author\_id odnoszący się do jedynego autora albumu z tabeli User (Relacja typu OneToMany)

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| id | Klucz główny, integer | Identyfikator |
| name | varchar | Nazwa albumu |
| description | text | Opis |
| created | datetime | Data utworzenia |
| updated | datetime | Data ostatniej aktualizacji |
| author\_id | Klucz obcy, integer | Id autora albumu |

### Tabela AlbumPhoto

Tabiera zawierająca informację o zdjęciach przypisanych do albumu. Jest to relacja typu ManyToMany ze względu na fakt że każde zdjęcie może mieć wiele albumów

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| id | Klucz główny, integer | Identyfikator |
| album\_id | Klucz obcy, integer | ID albumu |
| photo\_id | Klucz obcy, integer | ID zdjęcia |

### Tabela Tag

Tabela słownikowa zawierające unikalne nazwy Tagów.

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| id | Klucz główny, integer | Identyfikator |
| name | varchar, unikalne | Nazwa tagu |

### Tabela TagPhoto

Tabiera zawierająca informację o tagach przypisanych do zdjęć. Jest to relacja typu ManyToMany ze względu na fakt że każde zdjęcie może mieć wiele tagów

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| id | Klucz główny, integer | Identyfikator |
| tag\_id | Klucz obcy, integer | ID tagu |
| photo\_id | Klucz obcy, integer | ID zdjęcia |

### Tabela Favourite

Tabiera zawierająca informację o zdjęciach dodanych do ulubionych. Jest to relacja typu ManyToMany ze względu na fakt że różni użytkownicy mogą dodać jedne zdjęcie do ulubionych

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| id | Klucz główny, integer | Identyfikator |
| photo\_id | Klucz obcy, integer | ID zdjęcia |
| user\_id | Klucz obcy, integer | ID użytkownika |

# Implementacja warstwy logicznej

## Filtry routingu

Dostęp do poszczególnych stron aplikacji możliwy jest w zależności od stanu użytkownika. Aplikacja po stronie front-endu weryfikuje czy użytkownik jest zweryfikowany i na tej podstawie udostępnia użytkownikowi wejście do poszczególnych stron.

**Wydzielona grupa Route’ów przeznaczona dla zalogowanych i niezalogowanych użytkowników.**

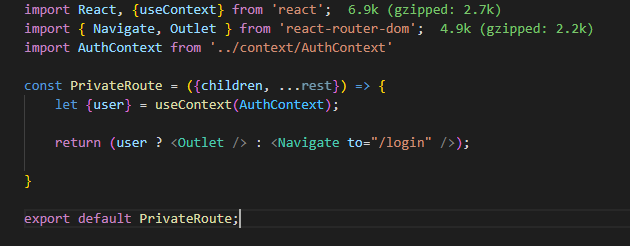
### PrivateRoute

Grupa routów przeznaczona dla zalogowanego użytkownika.

Obraz zawierający tekst, urządzenie

Opis wygenerowany automatycznie

### Kod PrivateRoute



### OnlyAnonymousRoute

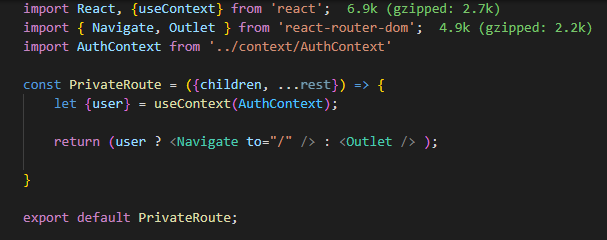
Grupa Route’ów przeznaczona tylko dla niezalogowanego użytkownika – pod nią znajdują się strony do logowania i rejestracji.

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Kod OnlyAnonymousRoute

W momencie gdy istnieje zmienna kontekstowa przechowująca dane o zalogowanym użytkowniku, mechanizm odsyła użytkownika na stronę główną. W innym wypadku, za pomocą elementu <Outlet />, wpuszcza użytkownika pod umieszczone w nim Route’y.



## Logowanie

Dostęp do strony logowania mają osoby, które nie posiadają aktualnie żadnej sesji. Weryfikacja tego stanu odbywa się w routingu po stronie front-endu za pomocą serwisu do obsługi sesji użytkowników (UserContext)

Użytkownik wypełnia formularz logowania, który następnie przekazywany jest do jednej z funkcji serwisu AuthContext, który wysyła dane z formularza do serwera backendu. Backend weryfikuje otrzymane dane i podejmuje odpowiednie działanie w zależności od tego czy użytkownik istnieje w systemie.

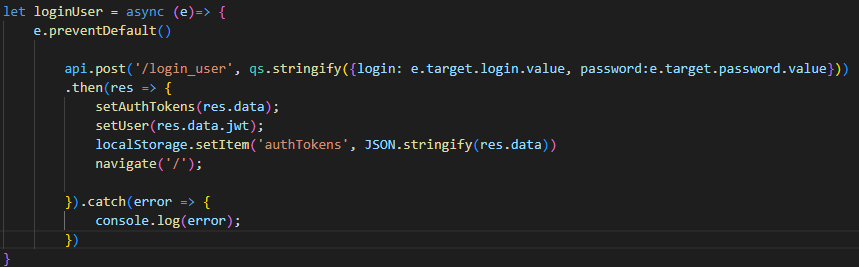
### Formularz logowania

Obraz zawierający tekst, wewnątrz, zrzut ekranu, ekran

Opis wygenerowany automatycznie

### Funkcja loginUser

Wszystkie dane z formularza znajdują się w argumencie funkcji e (oznaczającej event).



W przypadku poprawnego zalogowania wykonywane są następujące czynności:

* Otrzymane dane z tokenem zalogowania zapisywane są w LocalStorage przeglądarki,
* Dane o użytkowniku zapisywane są w Local Storage przeglądarki,
* Użytkownik odsyłany jest na stronę główną.

### Logowanie po stronie backendu

Za autentykacje / logowanie, odpowiada funkcja LoginView. Sprawdza ona czy istnieje użytkownik o podanych poświadczeniach, a następnie ustanawia sesje logowania w plikach cookie oraz zwraca dane użytkownika. W przeciwnym razie, funkcja ta zwróci do nadawcy błąd.

Obraz zawierający tekst

Opis wygenerowany automatycznie

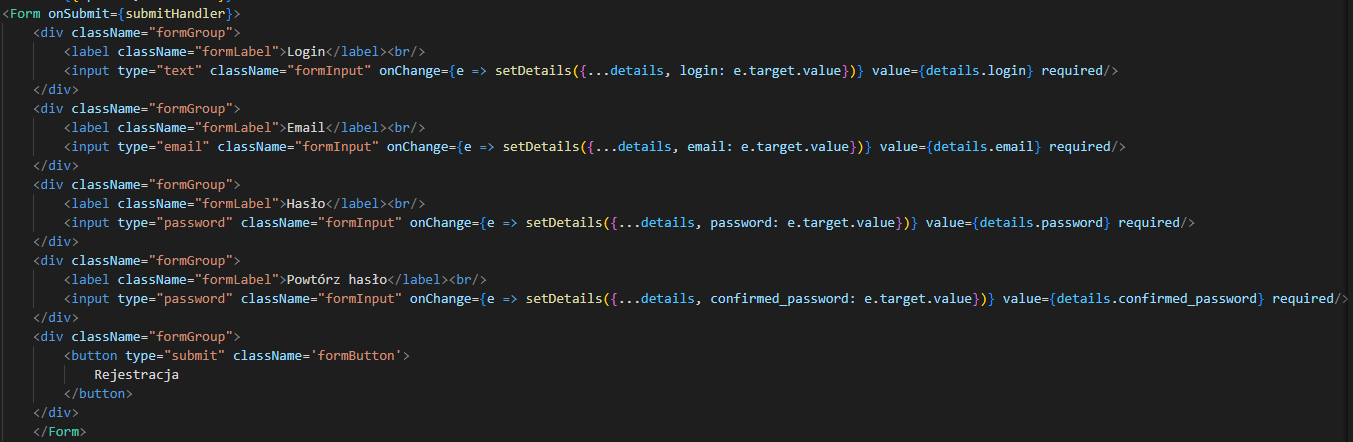
## Rejestracja

Dostęp do tej lokalizacji aplikacji jest również filtrowany wcześniej przez mechanizm do weryfikacji czy użytkownik jest anonimowy. Po poprawnej weryfikacji przez OnlyAnonymousRoute, użytkownikowi wyświetlana jest strona z formularzem.

Po stronie front-endu weryfikowane jest to czy wszystkie pola zostały wypełnione, oraz czy obydwa hasła są takie same.

Dane po weryfikacji wysyłane są do back-endu.

### Formularz rejestracji



### Funkcja submitHandler

W przypadku prawidłowej rejestracji, użytkownik odsyłany jest do strony logowania. W innym wypadku, uruchamiane jest okno dialogowe z informacją o błędzie zwróconą z back-endu.

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Rejestracja po stronie back-endu

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Dodawanie zdjęcia

Dodawanie zdjęcia odbywa się z poziomu galerii. Użytkownik na liście swoich zdjęć ma możliwość dodania zdjęcia za pomocą przycisku „Dodaj”.

Po wybraniu opcji „Dodaj” użytkownikowi na oknie dialogowym wyświetlany jest formularz do dodawania zdjęcia.

Formularz jest wstępnie weryfikowany na wypełnienie wszystkich pól za pomocą atrybutu *required*. Następnie dane wysyłane są do back-endu w celu ich dodania.  
Użytkownik na etapie dodawania zdjęcia, może dodać album oraz tagi przypisane do zdjęcia, bądź skorzystać z już istniejących.

### Formularz dodawania zdjęcia

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Funkcja addPhoto

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Dodawanie zdjęcia po stronie backendu

Po otrzymaniu zwalidowanych po stronie front-endu danych, jeśli jest to konieczne, za pomocą transakcyjności, dodawany jest szereg encji, m.in:

* Zdjęcie,
* Tagi,
* Albumy.

Zdjęcie jako plik, dodawane jest do Azure Blob Storage. W bazie danych natomiast zdjęcie identyfikowane jest po nazwie pliku.

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Edycja zdjęcia

Edycja zdjęcia odbywa się również z poziomu galerii użytkownika, który jest jej właścicielem. Po wybraniu opcji *Edytuj* pokazywane jest okno dialogowe, w którym użytkownik wypełnia formularz z już wypełnionymi, dotychczasowymi danymi.

Na poziomie front-endu weryfikowane jest wypełnienie wszystkich pól. Back-end po odebraniu danych z formularza podejmuje odpowiednie czynności.

Edycji ulegają również album oraz tagi dołączone do edytowanego zdjęcia. Oba te elementy można usunąć, dodać, bądź zaktualizować.

### Formularz edycji zdjęcia

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Funkcja editPhoto

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Edycja zdjęcia po stronie back-endu

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Dodawanie albumu

Dodawanie albumu odbywa się z poziomu widoku galerii użytkownika, który jest jej właścicielem. Użytkownikowi po wybraniu opcji *Dodaj* na wysokości albumów wyświetlany jest formularz, w którym może dodać album.

Należy wprowadzić nazwę albumu oraz jego opis, aby móc dodać nowy album.

### Formularz dodawania albumu

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Funkcja addAlbum

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Dodawanie albumu po stronie back-endu

Obraz zawierający tekst

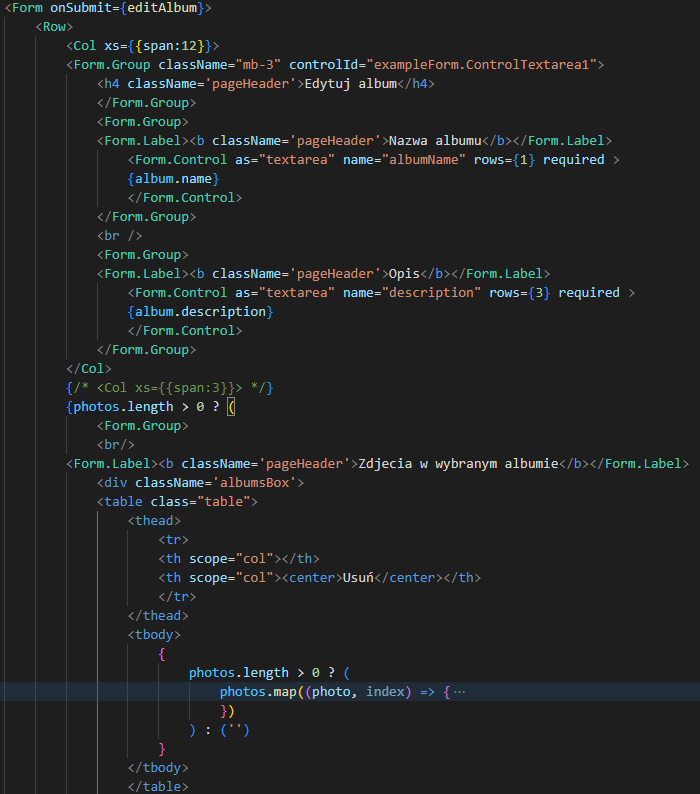
Opis wygenerowany automatycznie

## Edycja albumu

Użytkownik edytując album może zmienić jego nazwę lub opis. Dodatkową funkcjonalnością jest usunięcie wybranych, znajdujących się tam zdjęć.

Po wybraniu opcji *Edytuj* z poziomu widoku albumu, użytkownikowi wyświetlany jest formularz, w którym może wykonać wszystkie operacje.

### Formularz edycji albumu



### Funkcja editAlbum

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Edycja albumu po stronie back-endu

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Dodawanie tagów

Nowych i istniejących tagów odbywa się z poziomu dodawania / edycji zdjęcia z uwagi na wyłączne powiązanie obydwu obiektów.

Również z poziomu back-endu tagi są tworzone, jeżeli do danego zdjęcia przypisany został nieistniejący w bazie tag o danej nazwie.

### Dodawanie tagu/tagów po stronie backendu

Na początku sprawdzane jest, czy przesłane informacje o tagach nie są puste. Jeżeli nie, następnie dla każdego przesłanego taga sprawdza się ich obecność w już istniejących tagach. Jeżeli dany tag nie istnieje, tworzona jest nowa encja tagu.

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Dodawanie zdjęcia jako ulubione

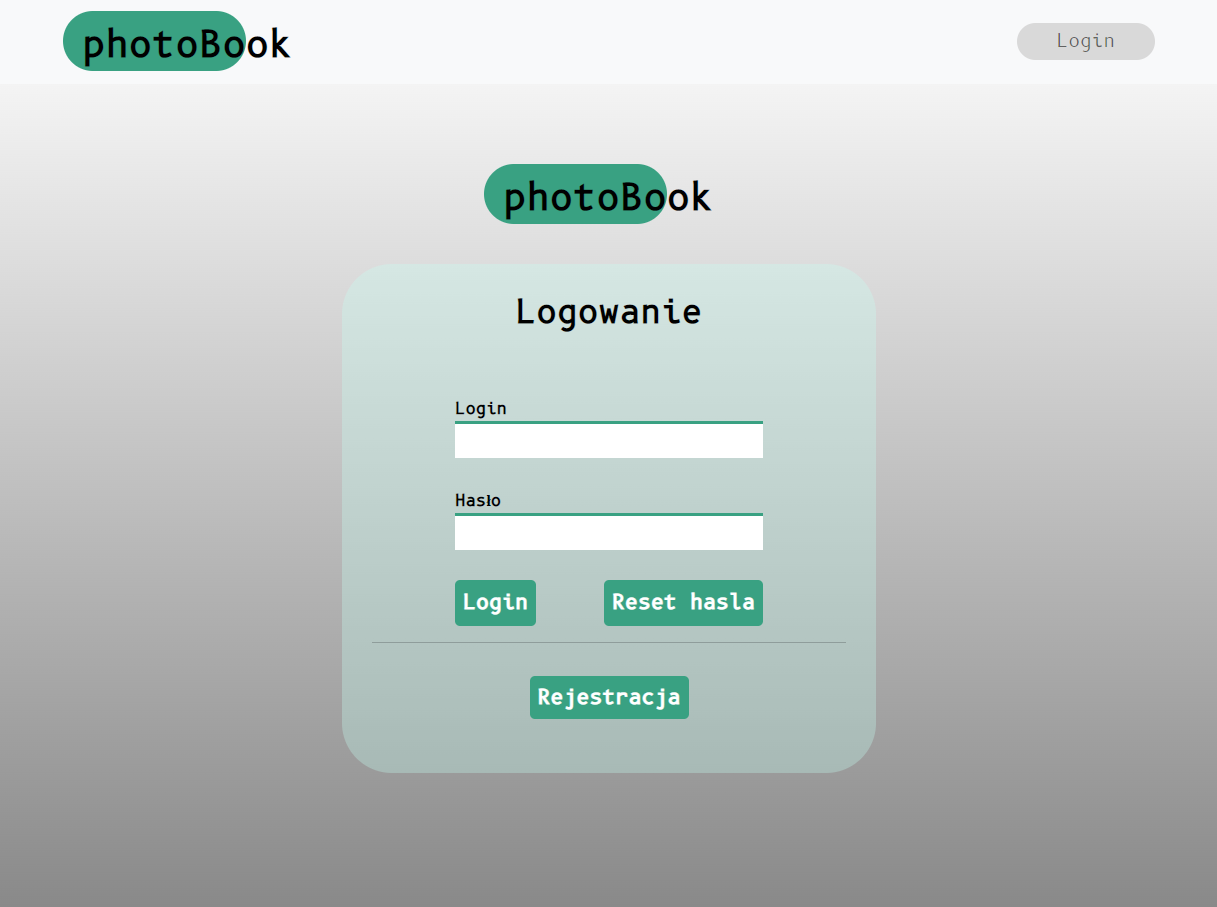
Zalogowany użytkownik ma możliwość oznaczania zdjęć jako ulubione. Dzięki temu polubione zdjęcie widnieć będzie na liście ulubionych zdjęć na stronie głównej.

Dodawanie zdjęcia jako ulubione odbywa się za pomocą kliknięcia gwiazdki przy tytule przeglądanego zdjęcia.

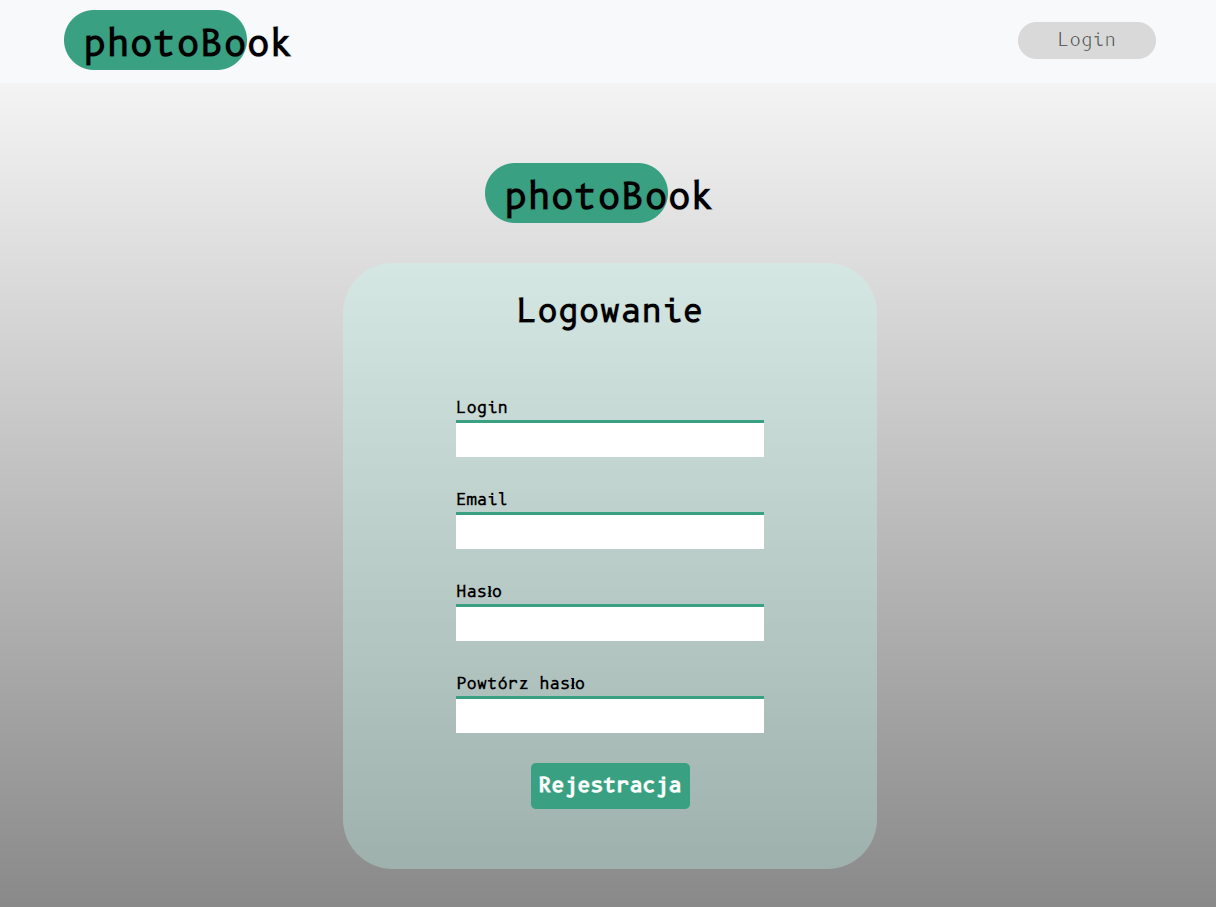
### Oznaczanie zdjęcia jako ulubione z poziomu front-endu

# Implementacja GUI

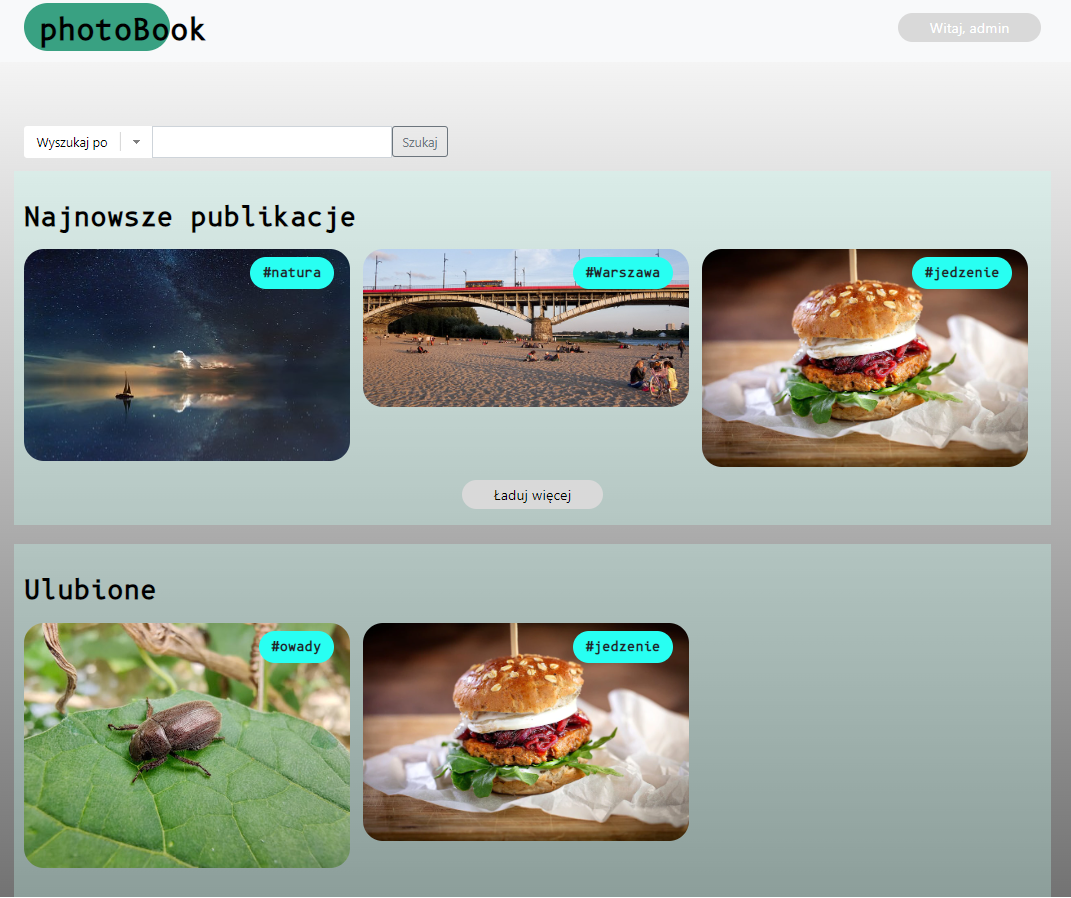
## Strona logowania



## Strona rejestracji



## Strona główna



## Zdjęcia

### Podgląd zdjęcia

Podgląd zdjęcia różni się w zależności od tego, czy użytkownik jest właścicielem danego zdjęcia. Jeżeli tak – wyświetlone zostaną dodatkowe opcje pozwalające na usunięcie lub edycje zdjęcia.

Obraz zawierający tekst, zrzut ekranu, monitor

Opis wygenerowany automatycznie

### 3.4.2 Dodawanie zdjęcia



### 3.4.3. Edycja zdjęcia



### 3.4.4. Usuwanie zdjęcia

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Albumy

### Podgląd albumu

Obraz zawierający tekst, zrzut ekranu

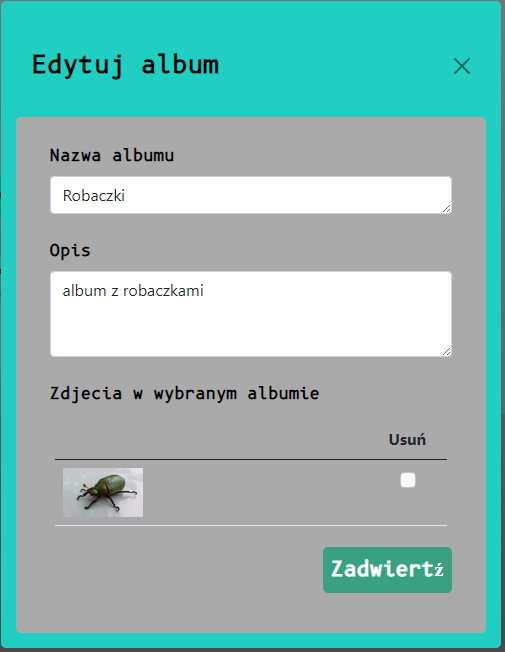
Opis wygenerowany automatycznie

### 3.4.2. Dodawanie albumu

Obraz zawierający tekst

Opis wygenerowany automatycznie

### 3.4.3. Edycja albumu

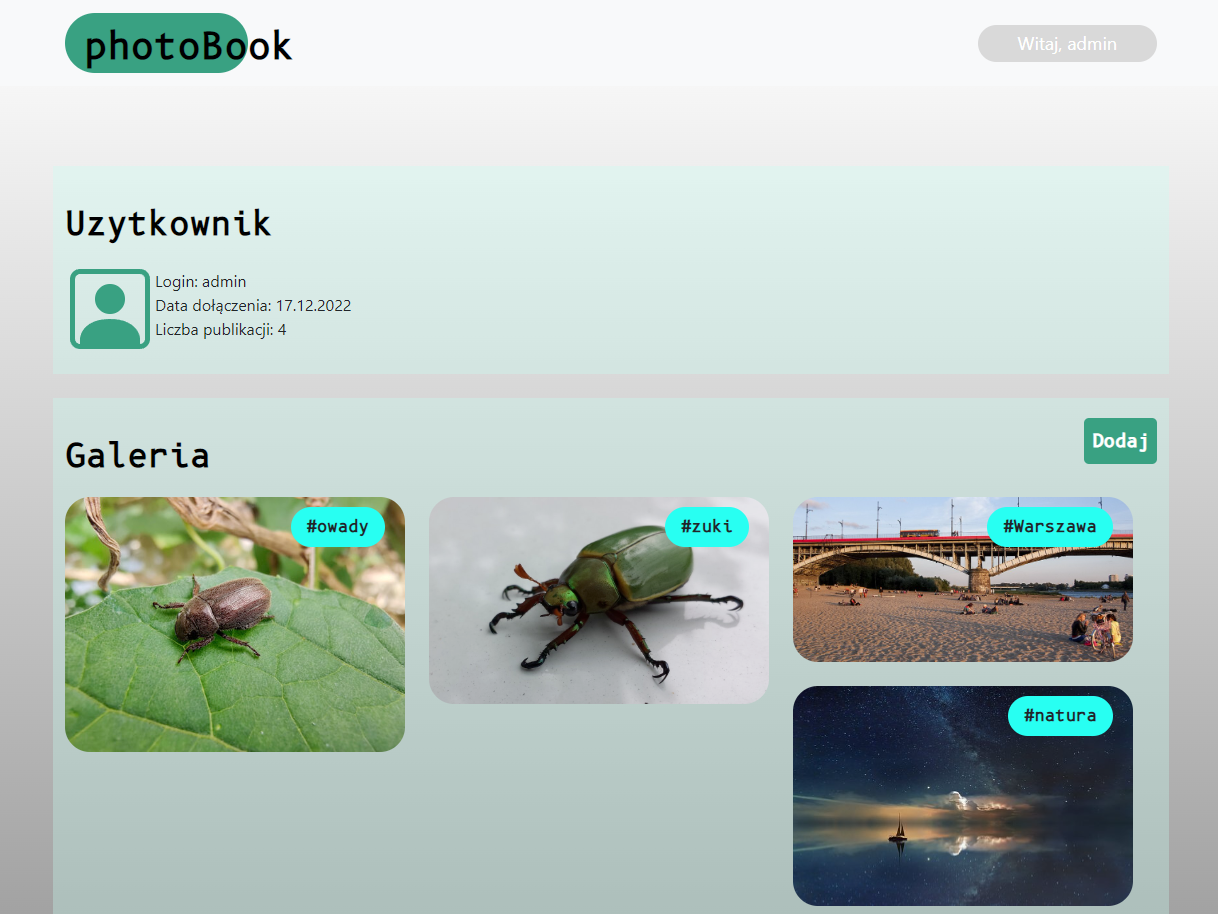


### 3.4.4. Usuwanie albumu

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Galeria / profil użytkownika



Obraz zawierający tekst, sprzęt elektroniczny, zrzut ekranu

Opis wygenerowany automatycznie

# Gotowy system informatyczny

Kod źródłowy dostępny jest na platformie github:

https://github.com/bchn7/projektZespolowy/tree/main/etap%204/aplikacja/backend

# Wyniki przeprowadzonych testów

Testy wykonywane będą przez platformę DataDog na już opublikowanej stronie.

## Testy funkcjonalności

### Zwrócenie przez API kodu 200 oraz listy zdjęć.

Konfiguracja test-case’u

Obraz zawierający tekst

Opis wygenerowany automatycznie

Wynik testu

Obraz zawierający stół

Opis wygenerowany automatycznie

### Test krokowy: dostęp do ulubionych zdjęć po zalogowaniu.

**Krok 1: logowanie.**

Aby stwierdzić, że logowanie przebiegło prawidłowo, API powinno zwrócić kod odpowiedzi 200, a body odpowiedzi powinno zawierać zmienną „jwt” zawierająca token.

Obraz zawierający tekst

Opis wygenerowany automatycznie

**Krok 2: ulubione zdjęcia użytkownika w odpowiedzi na wywołanie metody index.**

API zwraca tablicę favouritePhotos, jeżeli metodę index wywołuje zalogowany użytkownik.

Konfiguracja testu:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Wyniki testu

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Otrzymanie danych o zdjęciu.

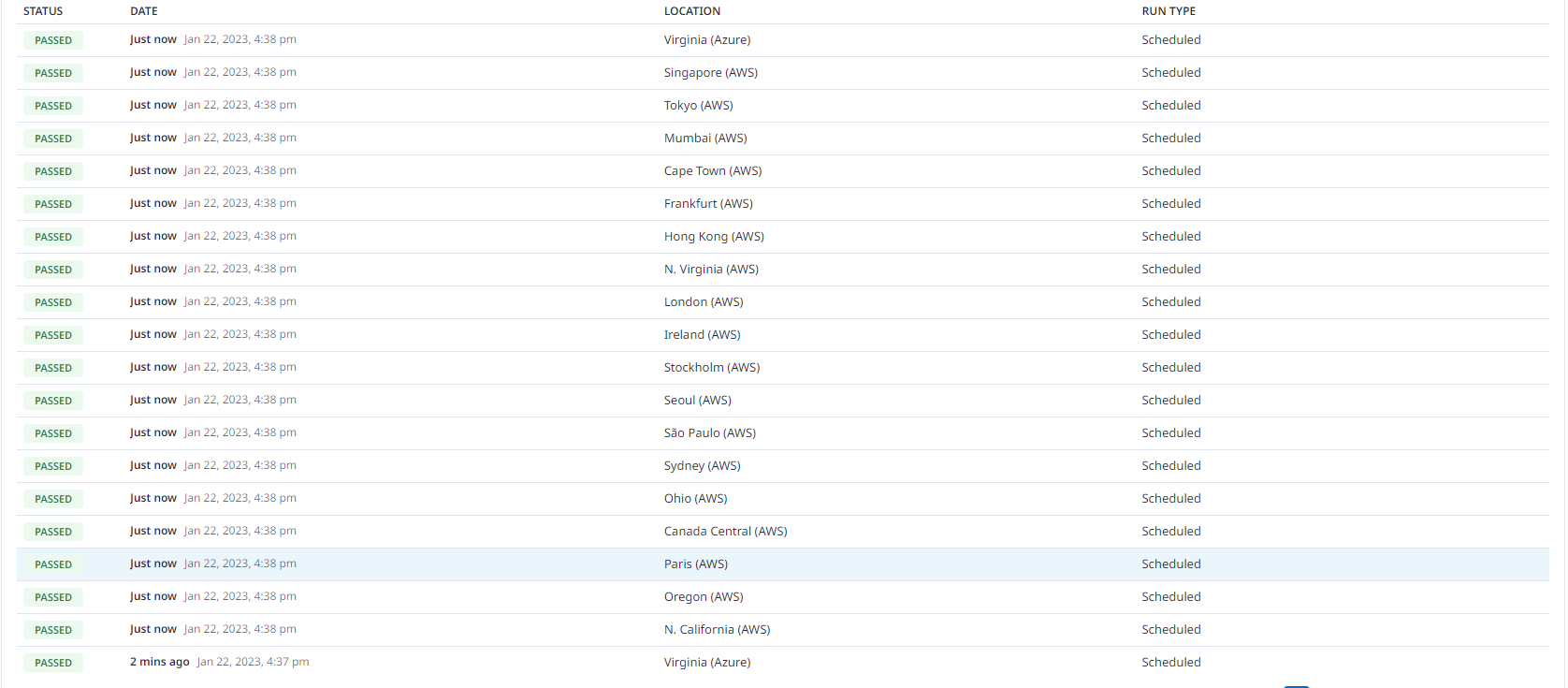
Po wywołaniu lokalizacji get\_photo/photo\_id w API, system powinien zwrócić dane o zdjęciu t.j.:

* Id zdjęcia,
* Nazwa zdjęcia,
* Dane o autorze,
* Tagi

Konfiguracja testu

## Testy obciążeniowe

Co jedną minutę wykonywane jest żądanie dostępu do domyślnej metody API, która powinna zwrócić najnowsze zdjęcia. Żądania te wykonywane są z ponad 20 lokalizacji na całym świecie. Jedynym domyślnym warunkiem oprogramowania DataDog, do przejścia testu pozytywnie jest kod 200 odpowiedzi oraz czas otrzymania odpowiedzi w rozsądnym czasie.



W ciągu jednej godziny nie wykryto żadnego niepowodzenia w testach obciążeniowych.

