

# Implementacja systemu i wykonanie testów sprawdzających

Etap IV | Grupa nr.3

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

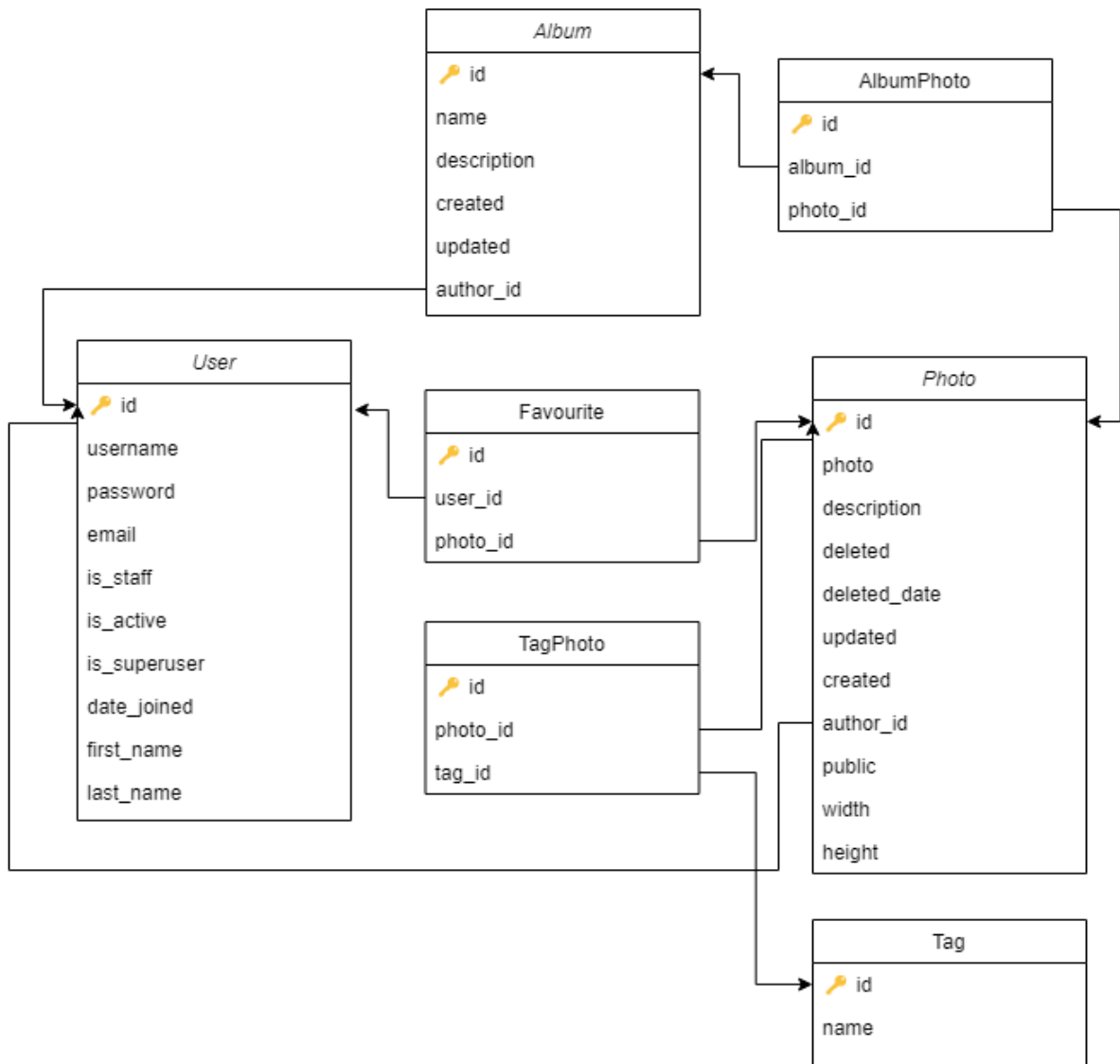
## Spis treści

1. Implementacja bazy danych.....	4
1.1. Diagram bazy danych .....	4
1.2. Sposób implementacji bazy danych .....	5
Tabela User .....	6
Tabela Photo .....	6
Tabela Album.....	7
Tabela AlbumPhoto .....	7
Tabela Tag.....	8
Tabela TagPhoto .....	8
Tabela Favourite .....	8
2. Implementacja warstwy logicznej .....	9
2.1. Filtry routingu .....	9
PrivateRoute .....	9
Kod PrivateRoute.....	9
OnlyAnonymousRoute .....	9
Kod OnlyAnonymousRoute .....	10
2.2. Logowanie .....	10
Formularz logowania.....	10
Funkcja loginUser .....	11
Logowanie po stronie backendu.....	11
2.3. Rejestracja.....	12
Formularz rejestracji .....	12
Funkcja submitHandler .....	12
Rejestracja po stronie back-endu .....	13
2.4. Dodawanie zdjęcia.....	13
Formularz dodawania zdjęcia.....	14
Funkcja addPhoto .....	15
Dodawanie zdjęcia po stronie backendu .....	15
2.5. Edycja zdjęcia.....	16
Formularz edycji zdjęcia.....	17
Funkcja editPhoto .....	17
Edycja zdjęcia po stronie back-endu .....	18
2.6. Dodawanie albumu .....	18
Formularz dodawania albumu .....	18

Funkcja addAlbum .....	19
Dodawanie albumu po stronie back-endu .....	19
2.7. Edycja albumu .....	20
Formularz edycji albumu .....	21
Funkcja editAlbum .....	21
Edycja albumu po stronie back-endu .....	22
2.8. Dodawanie tagów .....	23
Dodawanie tagu/tagów po stronie backendu .....	23
2.9. Dodawanie zdjęcia jako ulubione .....	23
Oznaczanie zdjęcia jako ulubione z poziomu front-endu .....	23
3. Implementacja GUI .....	24
3.1. Strona logowania .....	24
3.2. Strona rejestracji .....	25
3.3. Strona główna .....	26
3.4. Zdjęcia .....	26
3.4.1. Podgląd zdjęcia .....	27
3.4.2 Dodawanie zdjęcia .....	28
3.4.3. Edycja zdjęcia .....	29
3.4.4. Usuwanie zdjęcia .....	29
3.5. Albumy .....	31
3.5.1. Podgląd albumu .....	31
3.4.2. Dodawanie albumu .....	31
3.4.3. Edycja albumu .....	32
3.4.4. Usuwanie albumu .....	32
3.6. Galeria / profil użytkownika .....	32
4. Gotowy system informatyczny .....	33
5. Wyniki przeprowadzonych testów .....	33
Testy funkcjonalności .....	34
Zwrócenie przez API kodu 200 oraz listy zdjęć. ....	34
Test krokowy: dostęp do ulubionych zdjęć po zalogowaniu .....	34
Otrzymanie danych o zdjęciu. ....	35
Testy obciążeniowe .....	35

# 1. Implementacja bazy danych

## 1.1. Diagram bazy danych



## 1.2. Sposób implementacji bazy danych

```
from django.db import models
from django.contrib.auth.models import User

class Tag(models.Model):
    name = models.CharField(max_length=50)

    def __str__(self) -> str:
        return self.name

class Photo(models.Model):
    photo = models.ImageField()
    description = models.TextField(max_length=300, blank=True)
    tags = models.ForeignKey(Tag, on_delete=models.CASCADE)
    author = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name="photo_author"
    )
    favourite = models.ManyToManyField(User)
    deleted = models.BooleanField(default=False)
    deleted_date = models.DateTimeField(null=True)
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)
    public = models.BooleanField(default=False)
    width = models.IntegerField(blank=True)
    height = models.IntegerField(blank=True)

    def __str__(self):
        return f"{self.author} at {self.created}"

class Album(models.Model):
    name = models.CharField(max_length=50)
    description = models.TextField(max_length=300, blank=True)
    favourite = models.ManyToManyField(User)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    photos = models.ManyToManyField(Photo)
    author = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name="album_author"
    )

    def __str__(self):
        return self.name
```

Baza została zaimplementowana z użyciem DjangoORM. Tabele są reprezentowane 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 dodaje 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	bool	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 się 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 jednego 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

Tabela 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 jedno 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



## 2. Implementacja warstwy logicznej

### 2.1. 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.

```
<Route exact path="/" element={<PrivateRoute/>}>
  <Route path="/wyloguj" element={<Logout/>} />
</Route>
```

#### Kod PrivateRoute

```
import React, {useContext} from 'react'; 6.9k (gzipped: 2.7k)
import { Navigate, Outlet } from 'react-router-dom'; 4.9k (gzipped: 2.2k)
import AuthContext from '../context/AuthContext'

const PrivateRoute = ({children, ...rest}) => {
  let {user} = useContext(AuthContext);

  return (user ? <Outlet /> : <Navigate to="/login" />);
}

export default PrivateRoute;
```

#### OnlyAnonymousRoute

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

```
<Route exact path="/" element={<OnlyAnonymousRoute/>}>
  <Route path="/login" element={<Login />}></Route>
  <Route path="/rejestracja" element={<Register />}></Route>
</Route>
```

## 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.

```
import React, {useContext} from 'react'; 6.9k (gzipped: 2.7k)
import { Navigate, Outlet } from 'react-router-dom'; 4.9k (gzipped: 2.2k)
import AuthContext from '../context/AuthContext'

const PrivateRoute = ({children, ...rest}) => {
  let {user} = useContext(AuthContext);

  return (user ? <Navigate to="/" /> : <Outlet /> );
}

export default PrivateRoute;
```

## 2.2. 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

```
<Form onSubmit={loginUser}>
  <div className="formGroup">
    <label className="formLabel">Login</label><br/>
    <input type="text" className="formInput" name="login" onChange={e => setDetails({...details, login: e.target.value})} value={details.login} />
  </div>
  <div className="formGroup">
    <label className="formLabel">Hasło</label><br/>
    <input type="password" className="formInput" name="password" onChange={e => setDetails({...details, password: e.target.value})} value={details.password} />
  </div>
  <div className="formGroup">
    <button type="submit" className="formButton left">
      Login
    </button>
    <Link className="formButton right" to="/rejestracja">Reset hasła</Link>
  </div>
</Form>
```

## Funkcja loginUser

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

```
let loginUser = async (e) => {
  e.preventDefault()

  api.post('/login_user', qs.stringify({login: e.target.login.value, password: e.target.password.value}))
    .then(res => {
      setAuthTokens(res.data);
      setUser(res.data.jwt);
      localStorage.setItem('authTokens', JSON.stringify(res.data))
      navigate('/');
    }).catch(error => {
      console.log(error);
    })
}
```

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

```
class LoginView(APIView):
    def post(self, request):
        username = request.data['login']
        password = request.data['password']

        user = authenticate(username=username, password=password)

        if user is not None:
            payload = {
                'id': user.id,
                'username': user.username,
                'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=60),
                'iat': datetime.datetime.utcnow()
            }

            token = jwt.encode(payload, 'secret', algorithm='HS256')

            response = Response()

            response.set_cookie(key='jwt', value=token, httponly=True)

            response.data = {
                'jwt': token,
                'username': user.username,
                'userId': user.id
            }

            return response

        response = JsonResponse({"message": "Wystąpił błąd w trakcie logowania."})
        response.status_code = 500

        return response
```

## 2.3. 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

```
<Form onSubmit={handleSubmit}>
  <div className="formGroup">
    <label className="formLabel">Login</label><br/>
    <input type="text" className="formInput" onChange={e => setDetails({...details, login: e.target.value})} value={details.login} required/>
  </div>
  <div className="formGroup">
    <label className="formLabel">Email</label><br/>
    <input type="email" className="formInput" onChange={e => setDetails({...details, email: e.target.value})} value={details.email} required/>
  </div>
  <div className="formGroup">
    <label className="formLabel">Hasło</label><br/>
    <input type="password" className="formInput" onChange={e => setDetails({...details, password: e.target.value})} value={details.password} required/>
  </div>
  <div className="formGroup">
    <label className="formLabel">Powtórz hasło</label><br/>
    <input type="password" className="formInput" onChange={e => setDetails({...details, confirmed_password: e.target.value})} value={details.confirmed_password} required/>
  </div>
  <div className="formGroup">
    <button type="submit" className="formButton">
      Rejestracja
    </button>
  </div>
</Form>
```

### Funkcja handleSubmit

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.

```

const register = async () =>{
  if (details.password !== details.confirmed_password){
    setModalMessage("Hasła muszą być takie same.")
    setShow(true);
  } else {
    api.post('register', qs.stringify(details))
      .then(res => {
        if(res.data.status_code === 200){
          navigate('/login');
        }else{
          setModalMessage(res.data.message);
          setShow(true);
        }
      }).catch(error => {
        setModalMessage(error.message);
        console.log("error");
        setShow(true);
      })
  }
}

const submitHandler = e => {
  e.preventDefault();
  console.log(details);
  register();
};

```

Rejestracja po stronie back-endu

```

def register_user(request):
    try:
        user = User(username=request.POST['login'], email=request.POST['email'])
        user.set_password(request.POST['password'])
        user.save()

        return JsonResponse({'status_code': 200})

    except Exception as e:
        response = JsonResponse({'message': "Nie można utworzyć użytkownika +" + str(e)})
        response.status_code = 500

        return response

```

## 2.4. 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

```
<Form onSubmit={addPhoto}>
  <Row>
    <Col xs={{span:12}}>
      <Form.Group className="mb-3" controlId="exampleForm.ControlTextarea1">
        <center>
          <label className='custom-file-upload formButton'>
            <small>Wybierz plik</small>
            <input type="file" className="fileImport"
              id="avatar" name="photoFile"
              accept="image/png, image/jpeg" onChange={handleFileChange} required />
          </label>
        </center>
      </Form.Group>
      <Form.Group>
        <Form.Label><b className='pageHeader'>Nazwa zdjęcia</b></Form.Label>
        <Form.Control as="textarea" name="photoName" rows={1} required />
      </Form.Group>
      <Form.Group></Form.Group>
      <Form.Group>
        <Form.Label><b className='pageHeader'>Opis</b></Form.Label>
        <Form.Control as="textarea" name="description" rows={3} required />
      </Form.Group>
      <Form.Group>
        <br/>
        <Form.Label><b className='pageHeader'>Tagi</b></Form.Label>
        <Form.Control as="textarea" name="tags" rows={1} />
      </Form.Group>
    </Col>
    <Col xs={{span:3}}>
      <Form.Group>
        <br/>
        <Form.Label><b className='pageHeader'>Wybierz album</b></Form.Label>
        <div className='albumsBox'>
          {
            (albums.length > 0) ? (
              albums.map((album) => {
                return(
                  <div class="form-check">
                    <input class="form-check-input" type="radio" name="album" value={album.id} id="flexRadioDefault1" />
                    <label class="form-check-label" for="flexRadioDefault1">
                      {album.name}
                    </label>
                  </div>
                )
              })
            ) : (<i>Nie posiadasz żadnych albumów.</i>)
          }
        </div>
      </Form.Group>
    </Col>
  </Row>
</Form>
```

## Funkcja addPhoto

```
const addPhoto = async(e) => {  
  
  e.preventDefault();  
  
  let photo = e.target.photoFile.value;  
  let description = e.target.description.value;  
  let photoName = e.target.photoName.value;  
  let tags = e.target.tags.value;  
  let selectedAlbum = e.target.album.value;  
  let newAlbum = e.target.newAlbum.value;  
  
  if (photo){  
    setFile(e.target.photoFile.files[0]);  
    const formData = new FormData();  
    formData.append("name", photoName);  
    formData.append("description", description);  
    formData.append("tags", tags);  
    formData.append("photo", file);  
    formData.append("selectedAlbum", selectedAlbum);  
    formData.append("newAlbum", newAlbum);  
    formData.append("user_id", user.id);  
  
    try {  
      const response = await axios({  
        method: "post",  
        url: "http://localhost:8000/api/upload_photo",  
        data: formData,  
        headers: { "Content-Type": "multipart/form-data" },  
      }).then(res => {  
        setMessageInfo(res.data.message);  
        setShowInfo(true);  
        setIsLoading(false);  
        navigate('/galeria/'+user.id);  
      }).catch(errorResponse => {  
        setMessageInfo(errorResponse.data.message);  
        setShowInfo(true);  
      })  
  
    } catch(error) {  
      setMessageInfo(error);  
      setShowInfo(true);  
    }  
  }  
}
```

## 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.

```

def uploadPhoto(request):
    tags = request.POST['tags'].split(',')
    try:
        with transaction.atomic():
            user = User.objects.get(id=request.POST['user_id'])
            photo = Photo(photo=request.POST['name'] + ".jpg", title=request.POST['name'], description=request.POST['description'], author=user, public=True, deleted=False, width=1000, height=1000)
            photo.save()

            # handle tags
            for tag in tags:
                if tag != '':
                    existingTag = Tag.objects.filter(name=tag)
                    if existingTag.count() == 0:
                        existingTag = Tag(name=tag)
                        existingTag.save()
                        photo.tag.add(existingTag)
                    else:
                        photo.tag.add(existingTag[0])

            # handle albums
            if request.POST['selectedAlbum'] != '' or request.POST['newAlbum'] != '':
                if request.POST['selectedAlbum'] != '':
                    album = Album.objects.get(id=request.POST['selectedAlbum'])
                else:
                    album = Album(name=request.POST['newAlbum'], author=user)
                    album.save()

                photo.album.add(album)

            # upload to azure storage
            blobServiceClient = StorageBlob.BlobServiceClient(account_name='sqlvaugualab3vnpic', account_key='gT8yVwJANuTla7NPyFqCYah8rxmD/vUSeDSzKqCWNkx48cf8dZ4Us85coECVYPR6670PT4AStcIMsydc=')
            blobServiceClient.create_blob_from_bytes(container_name='photobook', blob_name=request.POST['name'] + ".jpg",
                                                    blob=request.FILES['photo'].read())

            return JsonResponse({'status_code': 200, 'message': 'Zdjęcie dodano pomyślnie'})

    except Exception as e:
        response = JsonResponse({'message': "Wystąpił problem ze wystawianiem zdjęcia: " + str(e)})
        response.status_code = 500
        print(e)
        return response

```

## 2.5. 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

```
<Form onSubmit={editPhoto}>
  <Row>
    <Col xs={{span:12}}>
      <Form.Group className="mb-3" controlId="exampleForm.ControlTextarea1">
        <center>
          </center>
        </Form.Group>
        <Form.Group>
          <Form.Label><b className='pageHeader'>Nazwa zdjęcia</b></Form.Label>
          <Form.Control as="textarea" name="photoName" rows={1} required >
            {photo.title}
          </Form.Control>
        </Form.Group>
        <Form.Group></Form.Group>
        <Form.Group>
          <Form.Label><b className='pageHeader'>Opis</b></Form.Label>
          <Form.Control as="textarea" name="description" rows={3} required >
            {photo.description}
          </Form.Control>
        </Form.Group>
        <Form.Group>
          <br/>
          <Form.Label><b className='pageHeader'>Tagi</b></Form.Label>
          <Form.Control as="textarea" name="tags" rows={1}>{tagNames}</Form.Control>
        </Form.Group>
      </Col>
      <Col xs={{span:3}}>
        <Form.Group>
          <br/>
          <Form.Label><b className='pageHeader'>Wybierz album</b></Form.Label>
          <div className='albumsBox'>...
          </div>
        </Form.Group>
      </Col>
    </Row>
    <br/>
    <Col xs={{span:4}}>
      <Form.Group>
        <br/>
        <Form.Label><b className='pageHeader'>Stworz nowy album</b></Form.Label>
        <Form.Control as="textarea" name="newAlbum" rows={1} placeholder="Podaj nazwę albumu"/>
      </Form.Group>
    </Col>
  </Form>
```

## Funkcja editPhoto

```
const editPhoto = async(e) => {
  e.preventDefault();

  let description = e.target.description.value;
  let photoName = e.target.photoName.value;
  let tags = e.target.tags.value;
  let selectedAlbum = e.target.album.value;
  let newAlbum = e.target.newAlbum.value;

  const formData = new FormData();
  formData.append("name", photoName);
  formData.append("description", description);
  formData.append("tags", tags);
  formData.append("selectedAlbum", selectedAlbum);
  formData.append("newAlbum", newAlbum);
  formData.append("user_id", user.id);

  try {
    const response = await axios({
      method: "post",
      url: "http://localhost:8000/api/edit_photo/" + photo.id,
      data: formData,
      headers: { "Content-Type": "multipart/form-data" },
    }).then(res => {
      getPhoto()
      setMessage(res.data.message);
      setShow(true);
      handleCloseEditPhoto();
    })
  } catch(error) {
    setMessage(error);
    setShow(true);
  }
}
```

## Edycja zdjęcia po stronie back-endu

```
def editPhoto(request, photo_id):
    photo = Photo.objects.get(id=photo_id)
    user = User.objects.get(id=request.POST['user_id'])

    # handle tags
    tags = request.POST['tags'].split(' ')

    with transaction.atomic():
        photo.tag.clear()
        if tags != ['']:
            for tag in tags:
                existingTag = Tag.objects.filter(name=tag)
                if existingTag.count() == 0:
                    existingTag = Tag(name=tag)
                    existingTag.save()
                    photo.tag.add(existingTag)
                else:
                    photo.tag.add(existingTag[0])

    # handle album
    photo.album.clear()
    if request.POST['selectedAlbum'] != '' or request.POST['newAlbum'] != '':
        if request.POST['selectedAlbum'] is not None:
            album = Album.objects.get(id=request.POST['selectedAlbum'])
        else:
            album = Album(name=request.POST['newAlbum'], author=user)
            album.save()

        photo.album.add(album)

    photo.description = request.POST['description']
    photo.title = request.POST['name']

    photo.save()

    return JsonResponse({"message": "Zmiany zostały zapisane pomyślnie."})
```

## 2.6. 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

```

<Form onSubmit={addAlbum}>
  <Form.Group className="mb-3" controlId="exampleForm.ControlTextarea1">
    <Form.Label><b className='pageHeader'>Nazwa albumu</b></Form.Label>
    <Form.Control as="textarea" name="albumName" rows={1} required />
  </Form.Group>
  <Form.Group className="mb-3" controlId="exampleForm.ControlTextarea1">
    <Form.Label><b className='pageHeader'>Opis</b></Form.Label>
    <Form.Control as="textarea" name="albumDescription" rows={2} required />
  </Form.Group>
  <input type="hidden" name="user_id" value={user_id} />
  <div className="formGroup">
    <button type="submit" className='formButton left'>
      Dodaj
    </button>
  </div>
</Form>

```

## Funkcja addAlbum

```

const addAlbum = async(e) => {
  e.preventDefault();
  api.post('/add_album', {
    name: e.target.albumName.value,
    description: e.target.albumDescription,
    user_id: user_id
  })
  .then(res => {
    setMessageInfo(res.data.message);
    setShowInfo(true);
    getUserAlbums();
    setIsLoading(false);
  }).catch(error => {
    setMessageInfo(error.message);
    setShowInfo(true);
    console.log(error)
  })
}

```

## Dodawanie albumu po stronie back-endu

```
def addAlbum(request):
    name = request.POST['name']
    description = request.POST['description']
    user_id = request.POST['user_id']

    try:
        user = User.objects.get(id=user_id)

        album = Album(name=name, description=description, author=user)
        album.save()

        return JsonResponse({'status_code': 200, 'message': "Album został dodany pomyślnie"})

    except DatabaseError as e:

        response = JsonResponse({'message': "Wystąpił błąd przy tworzeniu albumu"})
        response.status_code = 500

        return response
```

## 2.7. 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

```
<Form onSubmit={editAlbum}>
  <Row>
    <Col xs={{span:12}}>
      <Form.Group className="mb-3" controlId="exampleForm.ControlTextarea1">
        <h4 className='pageHeader'>Edytuj album</h4>
      </Form.Group>
      <Form.Group>
        <Form.Label><b className='pageHeader'>Nazwa albumu</b></Form.Label>
        <Form.Control as="textarea" name="albumName" rows={1} required >
          {album.name}
        </Form.Control>
      </Form.Group>
      <br />
      <Form.Group>
        <Form.Label><b className='pageHeader'>Opis</b></Form.Label>
        <Form.Control as="textarea" name="description" rows={3} required >
          {album.description}
        </Form.Control>
      </Form.Group>
    </Col>
    {/ * <Col xs={{span:3}}> */}
    {photos.length > 0 ? (
      <Form.Group>
        <br />
        <Form.Label><b className='pageHeader'>Zdjęcia w wybranym albumie</b></Form.Label>
        <div className='albumsBox'>
          <table class="table">
            <thead>
              <tr>
                <th scope="col"></th>
                <th scope="col"><center>Usuń</center></th>
              </tr>
            </thead>
            <tbody>
              {
                photos.length > 0 ? (
                  photos.map((photo, index) => { ...
                })
              ) : (``)
            }
          </tbody>
        </table>
      </Form.Group>
    ) : (``)}
  </Form>
```

## Funkcja editAlbum

```

const editAlbum = async(e) => {
  e.preventDefault();

  let albumName = e.target.albumName.value;
  let description = e.target.description.value;
  let photosToDelete = selectedPhotos;

  const formData = new FormData();
  formData.append("name", albumName);
  formData.append("description", description);
  formData.append("photosToDelete", photosToDelete);

  try {
    const response = await axios({
      method: "post",
      url: "http://localhost:8000/api/edit_album/" + album.id,
      data: formData,
      headers: { "Content-Type": "multipart/form-data" },
    }).then(res => {
      setMessage(res.data.message);
      setShow(true);
      getAlbumPhotos();
      handleCloseEditAlbum();
    }).catch(errorResponse => {
      setMessage(errorResponse.data.message);
      setShow(true);
    })

  } catch(error) {
    setMessage(error);
    setShow(true);
  }
}

```

Edycja albumu po stronie back-endu

```

def editAlbum(request, album_id):
    name = request.POST['name']
    description = request.POST['description']
    photosToDelete = request.POST['photosToDelete']

    album = Album.objects.get(id=album_id)

    if photosToDelete != '':
        for photo_id in photosToDelete.split(','):
            photo = Photo.objects.get(id=photo_id)
            photo.album.remove(album)

    album.name = name
    album.description = description
    album.save()

    return JsonResponse({'message': "Album zmodyfikowany prawidłowo."})

```

## 2.8. Dodawanie tagów

Nowych i istniejących tagów odbywa się z poziomu dodawania / edycji zdjęcia z uwagi na wyłączone 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.

```
for tag in tags:
    if tag != '':
        existingTag = Tag.objects.filter(name=tag)
        if existingTag.count() == 0:
            existingTag = Tag(name=tag)
            existingTag.save()
            photo.tag.add(existingTag)
        else:
            photo.tag.add(existingTag[0])
```

## 2.9. 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 przeglądane zdjęcia.

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

## 3. Implementacja GUI

### 3.1. Strona logowania

photoBook

Login

photoBook

Logowanie

Login

Hasło

Login

Reset hasła

Rejestracja



### 3.2. Strona rejestracji

photoBook

Login

photoBook

Logowanie

Login

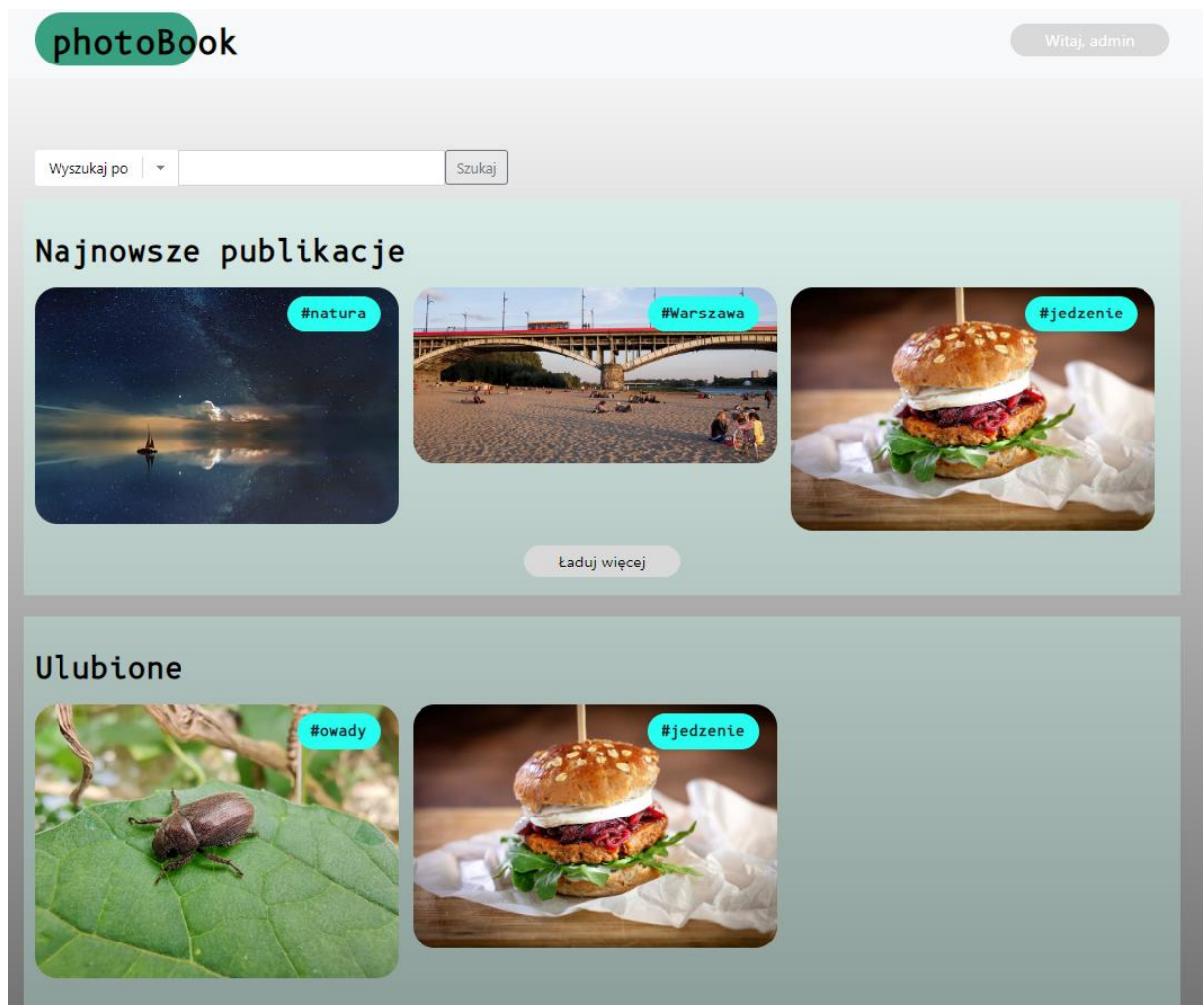
Email

Hasło

Powtórz hasło

Rejestracja

### 3.3. Strona główna




### 3.4. Zdjęcia

### 3.4.1. 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.

Noc ☆

  
admin

Dodano: 21.01.2023  
Zaktualizowano: 21.01.2023

Komentarze



Edytuj

Usuń

Tagi

#natura

#noc

Opis

Piękne zdjęcie przedstawiające noc

Brak komentarzy do tego zdjęcia.

Dodaj komentarz

Zatwierdź

### 3.4.2 Dodawanie zdjęcia

Dodaj zdjęcie

×

Wybierz plik

Nazwa zdjęcia

Opis

Tagi

Wybierz album

Stworz nowy album

☐ Robaczki

☐ Owady

☐ labe dzie

☐ Inne

Podaj

Dodaj

### 3.4.3. Edycja zdjęcia

## Edytuj zdjęcie

Nazwa zdjęcia

Vistula river

Opis

Wisla

Tagi

Wybierz album

☐ Robaczki

☐ Owady

☐ Iabedzie

☐ Inne

Stworz nowy album

Podaj

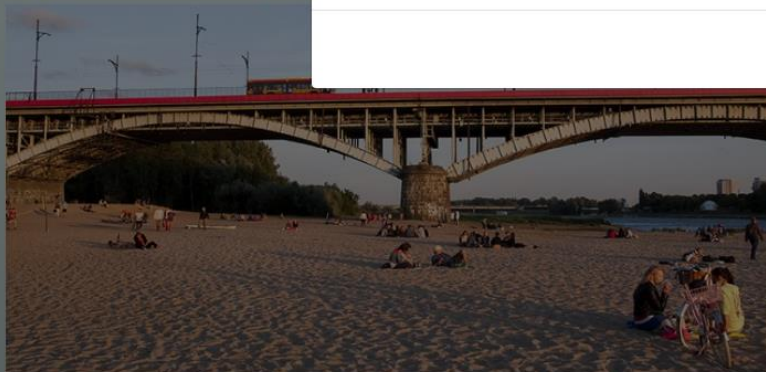
Dodaj

### 3.4.4. Usuwanie zdjęcia

# Vistula river



admin



Edytuj

Usuń

## Tagi

#Warszawa

## Opis

Wisła

### Usun zdjecie



Czy na pewno chcesz usunąć zdjęcie?

Tak

## omentarze

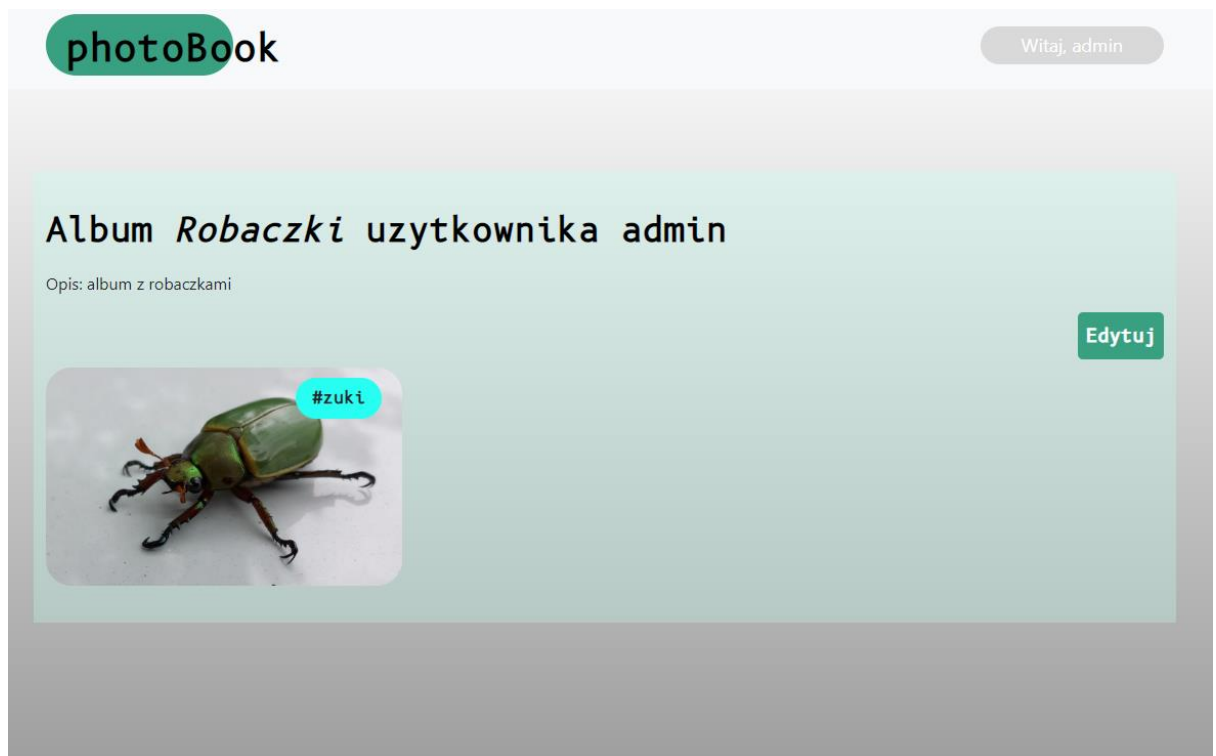
omentarzy do tego zdjęcia.

komentarz

Zatwierdź

## 3.5. Albumy

### 3.5.1. Podgląd albumu



### 3.4.2. Dodawanie albumu

The screenshot shows a modal window titled 'Dodaj album' with a close button (X) in the top right corner. The form contains two input fields: 'Nazwa albumu' and 'Opis'. Both fields are currently empty. At the bottom left of the form is a green 'Dodaj' button.

### 3.4.3. Edycja albumu

## Edytuj album

Nazwa albumu


Robaczki

Opis

album z robaczkami

Zdjęcia w wybranym albumie

Usun



Zadwierż

### 3.4.4. Usuwanie albumu

## Usun album

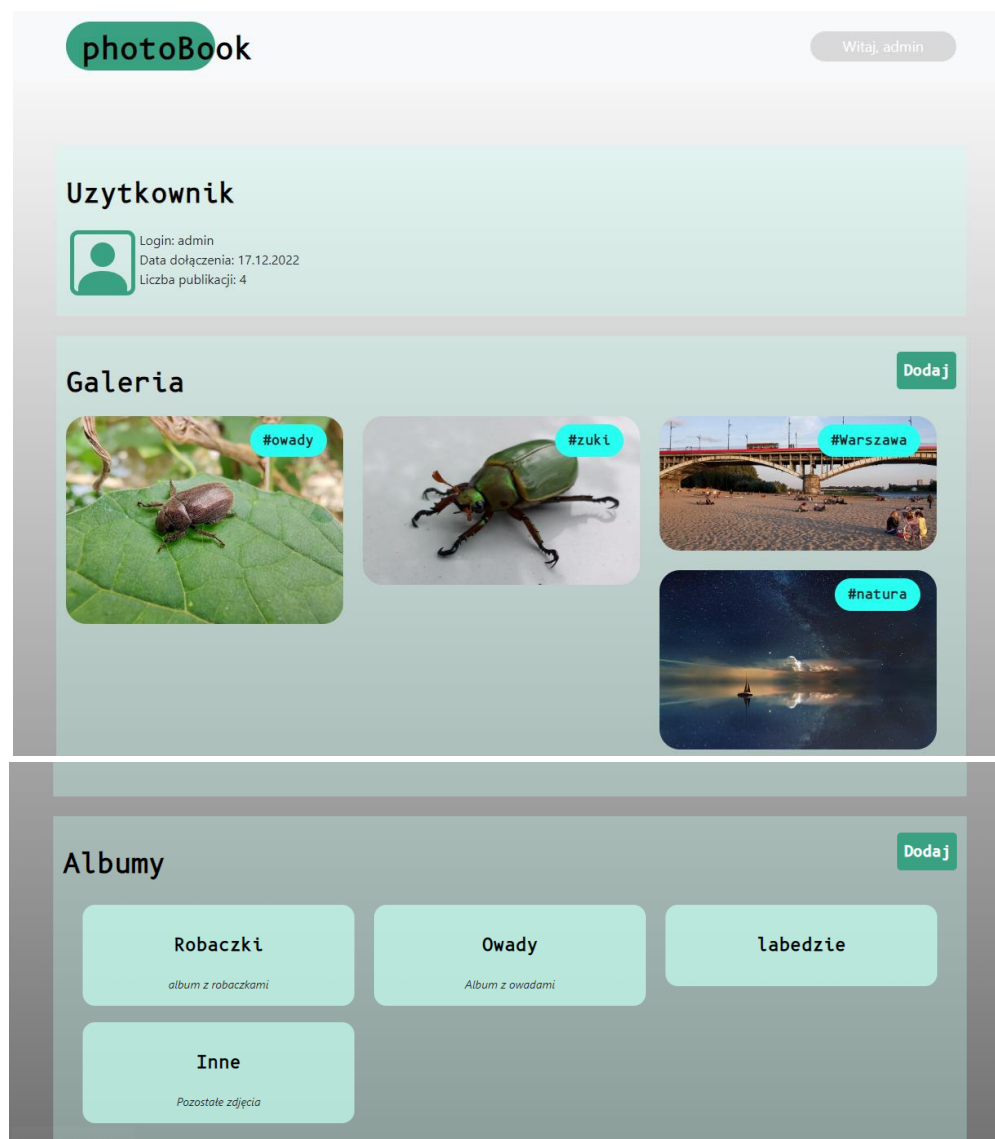
Czy na pewno chcesz usunąć album?

☒ Usun także wszystkie zdjęcia znajdujące się w albumie.

Tak

## 3.6. Galeria / profil użytkownika





## 4. Gotowy system informatyczny

Kod źródłowy dostępny jest na platformie github:

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

## 5. 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

Properties

HTTP Test

Test Id: e26-qu8-ke8

Created on Jan 22, 2023, 3:51 pm by Roberto Mayorga  
Modified on Jan 22, 2023, 3:51 pm by Roberto Mayorga

TAGS

You don't have any tags. Add tags to help organize your tests.

PRIORITY

Not Defined

OVERVIEW

GET <https://photobookapi.azurewebsites.net/api/> from 19 locations every 1 minute

2 assertions have been configured:

- Status Code should be 200
- Body should contain photos

MONITOR

Name [Synthetics] Test on photobookapi.azurewebsites.net/api/

Message No message configured

CI/CD EXECUTION

Blocking If this test fails, block the CI/CD pipeline

Execution rules apply only to tests running in CI/CD pipelines. [Learn more about Synthetic CI/CD Configuration](#)

## Wynik testu

Test Runs

Events Test Runs

All Passed Failed

Showing 3 test runs in the past 1 hour for selected locations

Last updated less than a minute ago Refresh

STATUS	DATE	LOCATION	RUN TYPE
PASSED	Just now Jan 22, 2023, 3:52 pm	Paris (AWS)	Scheduled
PASSED	1 min ago Jan 22, 2023, 3:51 pm	Paris (AWS)	Manually triggered
PASSED	1 min ago Jan 22, 2023, 3:51 pm	Paris (AWS)	Scheduled

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” zawierającą token.

URL \*

POST [https://photobookapi.azurewebsites.net/api/login\\_user](https://photobookapi.azurewebsites.net/api/login_user)

Advanced Options (2 configured)

Request Options 1 Authentication Query Params Request Body 1 Proxy Privacy

Body Type application/json

Request Body

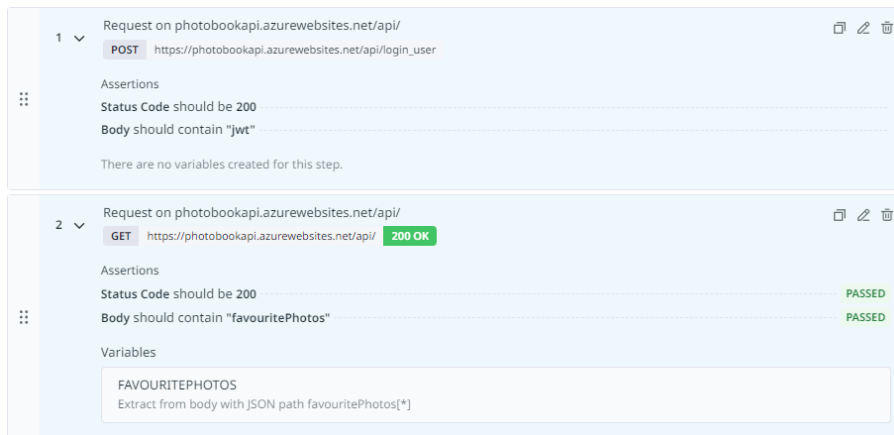
Type {{ to display available variables

```
1 {
2   "login": "admin",
3   "password": "123456",
4 }
```

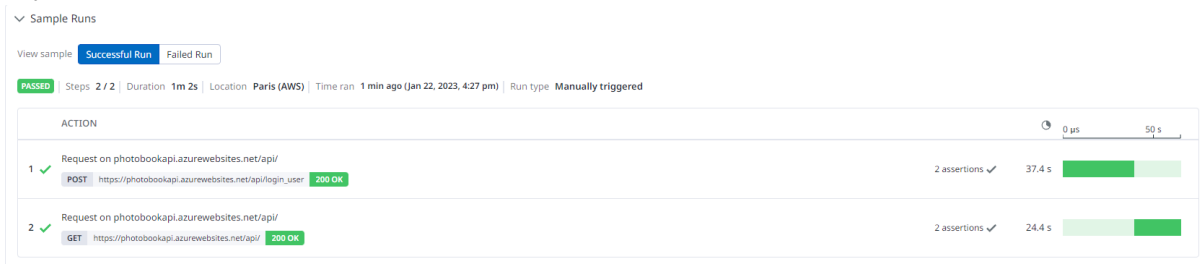
### Krok 2: ulubione zdjęcia użytkownika w odpowiedzi na wywołanie metody index.

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

Konfiguracja testu:



## Wyniki testu



## 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.

STATUS	DATE	LOCATION	RUN TYPE
PASSED	Just now Jan 22, 2023, 4:38 pm	Virginia (Azure)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Singapore (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Tokyo (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Mumbai (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Cape Town (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Frankfurt (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Hong Kong (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	N. Virginia (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	London (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Ireland (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Stockholm (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Seoul (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	São Paulo (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Sydney (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Ohio (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Canada Central (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Paris (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	Oregon (AWS)	Scheduled
PASSED	Just now Jan 22, 2023, 4:38 pm	N. California (AWS)	Scheduled
PASSED	2 mins ago Jan 22, 2023, 4:37 pm	Virginia (Azure)	Scheduled

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


Test Runs

EventsTest Runs

AllPassedFailed

No test runs in the past 1 hour for selected locations

Lost updated less than a minute agoRefresh

STATUS	DATE	LOCATION	RUN TYPE
<div><div>No test runs yet</div></div>			