



Kolegium Nauk Przyrodniczych
Uniwersytet Rzeszowski

Przedmiot:

Bazy danych II

Tandem

Wykonał

Natan Czernicki, 131417

Dmytro Gnatyk, 120488

Prowadzący: Dr inż. Piotr Grochowalski

Rzeszów 2025

Specyfikacja tematu projektu

Przedstawienie projektowanej rzeczywistości

Projekt Tandem to wszechstronna aplikacja społecznościowa, która oferuje użytkownikom dynamiczną platformę do wyrażania siebie, komunikacji i budowania relacji. Główne funkcje Tandem obejmują możliwość tworzenia indywidualnych profili, gdzie użytkownicy mogą podawać swoje dane, wgrywać zdjęcia profilowe oraz opcjonalnie dodać opis o sobie. Każdy profil wyświetla liczbę obserwujących, liczbę polubień i liczbę opublikowanych postów, co pozwala użytkownikom lepiej zrozumieć swoje miejsce w społeczności.

Aplikacja obejmuje:

1. Profile użytkowników

Spersonalizowane profile z możliwością dodania informacji o sobie, zdjęcia profilowego i opcjonalnego opisu.

Statystyki dotyczące liczby obserwujących, polubień i łącznej liczby postów, które pomagają użytkownikom ocenić swoją obecność w społeczności.

2. Udostępnianie mediów

Możliwość udostępniania zdjęć, filmów i plików audio.

Posty multimedialne mogą zawierać podpisy, a użytkownicy mogą wchodzić w interakcje poprzez polubienia i komentarze, co zwiększa zaangażowanie.

3. Grupy

Grupy publiczne: Otwarte dla wszystkich użytkowników, umożliwiające swobodną dyskusję na tematy wspólnych zainteresowań.

Aspekt projektowy bazy danych

Struktura bazy danych

- **User:** tabela przechowująca informacje o użytkownikach platformy (login, nazwa użytkownika, adres e-mail, hasło, opis użytkownika, zdjęcie profilowe).
- **Follows:** tabela przechowująca informacje o relacjach między użytkownikami, gdzie użytkownicy mogą obserwować innych (ID obserwującego, ID obserwowanego).
- **Photo:** tabela przechowująca informacje o zdjęciach (ID zdjęcia, URL zdjęcia, opis, data publikacji, ID użytkownika).
- **Video:** tabela przechowująca informacje o filmach (ID filmu, URL filmu, opis, data publikacji, ID użytkownika).
- **Audio:** tabela przechowująca informacje o plikach audio (ID pliku audio, URL pliku, data publikacji, ID użytkownika).
- **Text:** tabela przechowująca treści tekstowe (ID treści, zawartość tekstowa, data publikacji, ID użytkownika).
- **Group:** tabela przechowująca informacje o grupach (ID grupy, nazwa, klucz grupy, ikona, opis, data utworzenia, typ grupy, kod dostępu, powiązane wiadomości).
group_messages: tabela relacyjna łącząca grupy z wiadomościami (ID grupy, ID wiadomości).
- **Optional:** tabela przechowująca członkostwo użytkowników w grupach i status admina (ID, ID użytkownika, ID grupy, admin).

Relacje

1. User (Użytkownik)

- **Relacje:**
 - Ma relację z tabelą **Group** za pomocą tabeli **Admins**, która określa, czy dany użytkownik jest administratorem grupy.
 - Ma relację z tabelą **follows**, gdzie użytkownicy mogą obserwować innych użytkowników (self-referencyjna relacja w tabeli follows).
 - Relacja z tabelą **Message** – użytkownik może wysyłać wiadomości jako nadawca (**sender** FK w tabeli Message).
 - Relacja z tabelą **Content** przez atrybut **content** (FK), który może przechowywać powiązane treści multimedialne użytkownika.
-

2. Group (Grupa)

- **Relacje:**
 - Relacja z tabelą **Admins** – określa użytkowników będących administratorami grupy.
 - Relacja z tabelą **Message** – grupa może być odbiorcą wiadomości (poprzez FK w tabeli Message).
 - Relacja z tabelą **Optional** – grupa może mieć przypisane dodatkowe treści multimedialne (zdjęcia, wideo, audio).
 - Użytkownicy mogą dołączać do grupy jako członkowie poprzez przypisanie w tabeli **Admins** lub przez mechanizm obserwacji (follows).
-

3. Message (Wiadomość)

- **Relacje:**
 - Relacja z tabelą **User** – wiadomość ma nadawcę, którego identyfikator jest zapisany jako **sender** (FK).
 - Relacja z tabelą **Group** – wiadomość może być wysyłana do grupy lub dotyczyć grupy.
 - Wiadomość zawiera treść tekstową.
-

4. Content (Treści multimedialne)

- **Relacje:**
 - Relacja z tabelami:
 - **Photo** (FK): Treść może być zdjęciem.
 - **Video** (FK): Treść może być wideo.
 - **Audio** (FK): Treść może być plikiem audio.
 - Relacja z tabelą **Optional** – pozwala przypisać różne treści do grupy.
-

5. Photo (Zdjęcie)

- **Relacje:**
 - Relacja z tabelą **Content** – zdjęcie jest rodzajem treści multimedialnej.
 - Zdjęcie może być udostępniane przez użytkownika lub przypisane do grupy.

6. Video (Wideo)

- **Relacje:**
 - Relacja z tabelą **Content** – wideo jest rodzajem treści multimedialnej.
 - Może być powiązane z grupą lub użytkownikiem przez tabelę **Optional**.
-

7. Audio

- **Relacje:**
 - Relacja z tabelą **Content** – audio jest rodzajem treści multimedialnej.
 - Może być powiązane z grupą przez tabelę **Optional**.
-

8. Optional (Opcjonalne treści grup)

- **Relacje:**
 - Relacja z tabelą **Group** – opcjonalne treści mogą być przypisane do grup.
 - Relacja z tabelą **Content** – opcjonalne treści mogą zawierać zdjęcia, wideo lub audio.
-

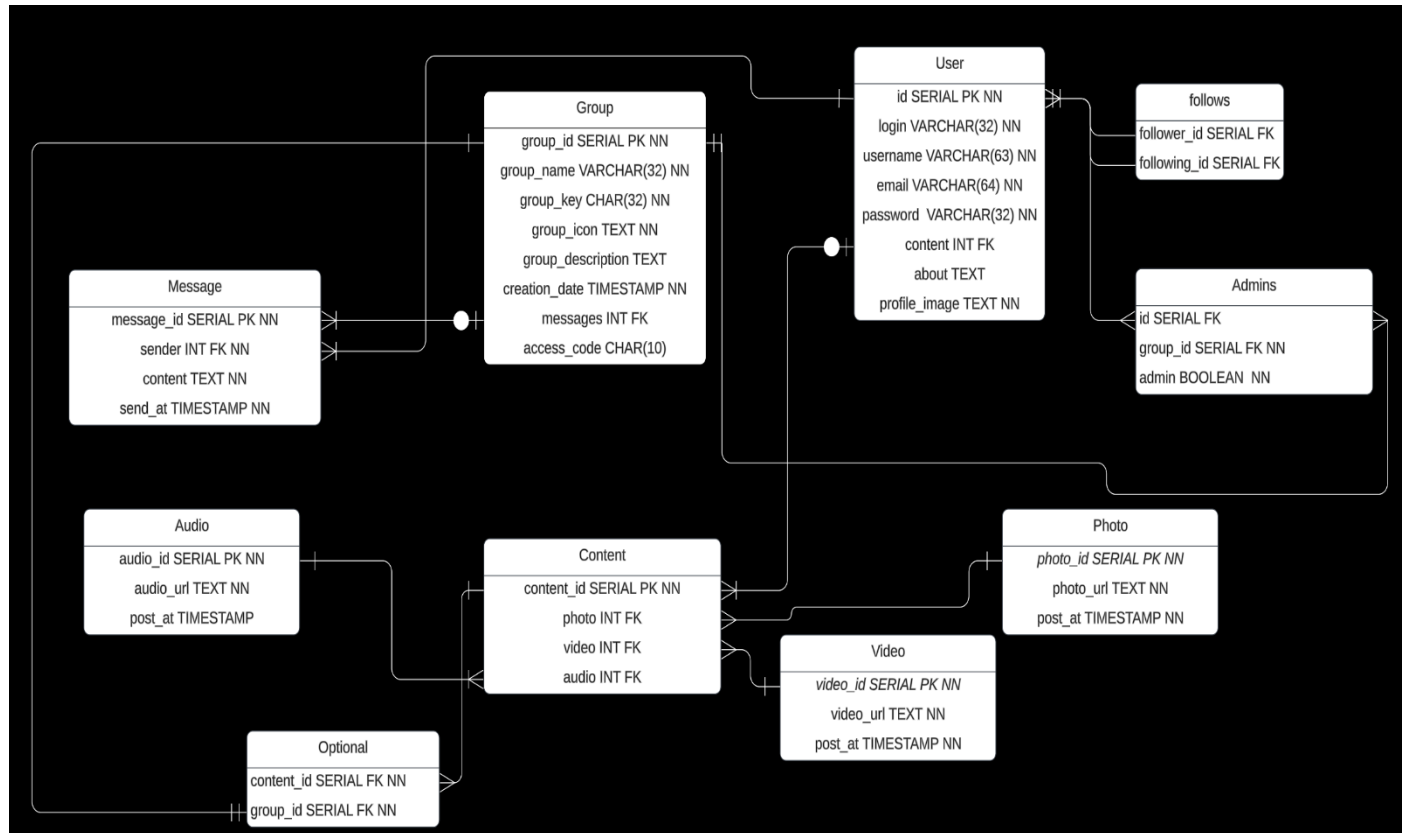
9. follows (Obserwacje między użytkownikami)

- **Relacje:**
 - Self-referencyjna relacja w tabeli **User** – użytkownicy mogą obserwować innych użytkowników.
 - Zawiera atrybuty **follower_id** i **following_id** jako FK wskazujące użytkowników.
-

10. Admins (Administratorzy grup)

- **Relacje:**
 - Relacja z tabelą **User** – określa, którzy użytkownicy są administratorami grupy.
 - Relacja z tabelą **Group** – przypisuje użytkowników jako administratorów grup.

Diagram ERD



Rysunek 1. Schemat ERD

Aspekt projektowy niezbędnych funkcjonalności

Procedury

- add_user: Dodaje nowego użytkownika z zahaszowanym hasłem.

```
CREATE OR REPLACE PROCEDURE user_management.add_user(  
  p_login VARCHAR(32),  
  p_username VARCHAR(63),  
  p_email VARCHAR(64),  
  p_password VARCHAR(32),  
  p_about TEXT,  
  p_profile_image TEXT  
) AS $$  
BEGIN  
  INSERT INTO user_management."User" (login, username, email,  
    password, about, profile_image)  
    VALUES (p_login, p_username, p_email, crypt(p_password, gen_salt('bf',  
      8)), p_about, p_profile_image);  
END;  
$$ LANGUAGE plpgsql;
```

- update_user_profile: Aktualizuje profil użytkownika (opis, zdjęcie profilowe, nazwa użytkownika).

```
CREATE OR REPLACE PROCEDURE user_management.update_user_profile(  
  p_id INT,  
  p_about TEXT,  
  p_profile_image TEXT  
) AS $$  
BEGIN  
  UPDATE user_management."User"  
    SET about = p_about,  
    profile_image = p_profile_image  
    WHERE id = p_id;  
END;  
$$ LANGUAGE plpgsql;
```

- add_user_to_group: Dodaje użytkownika do grupy.

```
CREATE OR REPLACE PROCEDURE group_management.add_user_to_group(
  p_user_id BIGINT,
  p_group_id BIGINT,
  p_admin BOOLEAN
) AS $$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM group_management."Group" WHERE
    group_id p_group_id) THEN
    RAISE EXCEPTION 'Group with ID % does not exist', p_group_id;
  END IF;
  INSERT INTO group_management.Optional (user_id, group_id, admin)
  VALUES (p_user_id, p_group_id, p_admin)
  ON CONFLICT (user_id, group_id) DO NOTHING;
END;
$$ LANGUAGE plpgsql;
```

- follow_user / unfollow_user: Dodawanie/usuwanie relacji między użytkownikami.

```
CREATE OR REPLACE PROCEDURE user_management.follow_user(
  p_follower_id BIGINT,
  p_following_id BIGINT
)
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO user_management.follows (followers, following)
  VALUES (p_follower_id, p_following_id)
  ON CONFLICT (followers, following) DO NOTHING;
END;
$$;

CREATE OR REPLACE PROCEDURE user_management.unfollow_user(
  p_follower_id BIGINT,
  p_following_id BIGINT
)
LANGUAGE plpgsql
AS $$
BEGIN
  DELETE FROM user_management.follows
  WHERE followers = p_follower_id AND following = p_following_id;
END;
$$;
```


Operacje CRUD

- **Create:**

Procedura group_management.create_group tworzy nową grupę z informacjami, takimi jak nazwa, opis, ikona i kod dostępu.

```
CREATE OR REPLACE PROCEDURE group_management.create_group(  
    p_group_name VARCHAR(32),  
    p_group_icon TEXT,  
    p_group_description TEXT,  
    p_access_code CHAR(10),  
    p_type BOOLEAN  
) AS $$  
BEGIN  
    INSERT INTO group_management."Group" (group_name, group_icon,  
    group_description, access_code, type)  
    VALUES (p_group_name, p_group_icon, p_group_description, p_access_code,  
    p_type);  
END;  
$$ LANGUAGE plpgsql;
```

- **Read:**

Funkcja group_management.list_group_users zwraca listę użytkowników w grupie.

```
CREATE OR REPLACE FUNCTION group_management.list_group_users(p_group_id  
BIGINT)  
RETURNS TABLE(id INT, username VARCHAR, admin BOOLEAN) AS $$  
BEGIN  
    RETURN QUERY  
    SELECT u.id, u.username, o.admin  
    FROM user_management."User" u  
    INNER JOIN group_management.Optional o ON u.id = o.user_id  
    WHERE o.group_id = p_group_id;  
END;  
$$ LANGUAGE plpgsql;
```

- **Update:**

Procedura group_management.update_group umożliwia aktualizację informacji grupy.

```
CREATE OR REPLACE PROCEDURE group_management.update_group(  
    p_group_id BIGINT,  
    p_group_icon TEXT,  
    p_group_description TEXT  
) AS $$  
BEGIN  
    IF NOT EXISTS (SELECT 1 FROM group_management."Group" WHERE group_id =  
    p_group_id) THEN
```

```

        RAISE EXCEPTION 'Group with ID % does not exist', p_group_id;
    END IF;

    UPDATE group_management."Group"
    SET group_icon = p_group_icon,
        group_description = p_group_description
    WHERE group_id = p_group_id;
END;
$$ LANGUAGE plpgsql;

```

- **Delete:**

Procedura group_management.delete_group usuwa grupę oraz powiązane dane, w tym członkostwa i zawartość.

```

CREATE OR REPLACE PROCEDURE group_management.delete_group(
    p_group_id BIGINT
) AS $$
BEGIN
    DELETE FROM group_management.Optional WHERE group_id = p_group_id;

    DELETE FROM content_management.GroupContent WHERE group_id =
p_group_id;

    DELETE FROM group_management.group_messages WHERE group_id =
p_group_id;

    DELETE FROM group_management."Group" WHERE group_id = p_group_id;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Group with ID % does not exist', p_group_id;
    END IF;
END;
$$ LANGUAGE plpgsql;

```

Funkcje

4. **user_management.get_follower_count**

- **Opis:** Funkcja zwraca liczbę osób, które obserwują wskazanego użytkownika (oznaczonego przez p_user_id).

```
CREATE OR REPLACE FUNCTION user_management.get_follower_count(  
    p_user_id BIGINT  
) RETURNS INT AS $$  
    BEGIN  
    RETURN (  
        SELECT COUNT(*)  
        FROM user_management.follows  
        WHERE following = p_user_id  
    );  
    END;  
    $$ LANGUAGE plpgsql;
```

2. **user_management.get_following_count**

- **Opis:** Funkcja zwraca liczbę osób, które są obserwowane przez wskazanego użytkownika (oznaczonego przez p_user_id).

```
CREATE OR REPLACE FUNCTION user_management.get_following_count(  
    p_user_id BIGINT  
) RETURNS INT AS $$  
    BEGIN  
    RETURN (  
        SELECT COUNT(*)  
        FROM user_management.follows  
        WHERE followers = p_user_id  
    );  
    END;  
    $$ LANGUAGE plpgsql;
```

3. **user_management.check_authorization**

Opis: Funkcja sprawdza poprawność danych logowania użytkownika, porównując podane hasło z przechowywanym w bazie danych.

```
CREATE OR REPLACE FUNCTION user_management.check_authorization(  
    p_login VARCHAR(32),  
    p_password VARCHAR(32)  
) RETURNS BOOLEAN AS $$  
    DECLARE  
        stored_password TEXT;  
    BEGIN  
        SELECT password INTO stored_password  
        FROM user_management."User"  
        WHERE login = p_login;  
        IF NOT FOUND OR crypt(p_password, stored_password) <>  
stored_password THEN  
            RETURN FALSE;  
        END IF;  
        RETURN TRUE;  
    END;  
    $$ LANGUAGE plpgsql;
```

5. **user_management.is_user_following**

Funkcja sprawdza, czy użytkownik o podanym ID (p_follower_id) obserwuje innego użytkownika o wskazanym ID (p_following_id).

```
CREATE OR REPLACE FUNCTION user_management.is_user_following(  
    p_follower_id BIGINT,  
    p_following_id BIGINT  
)  
    RETURNS BOOLEAN  
    LANGUAGE plpgsql  
AS $$  
    DECLARE  
        is_following BOOLEAN;  
    BEGIN  
        SELECT EXISTS(  
            SELECT 1 FROM user_management.follows  
            WHERE followers = p_follower_id AND following =  
p_following_id  
) INTO is_following;  
  
        RETURN is_following;  
    END;  
    $$;
```

1. **Kursor: user_management.get_following**

```
CREATE OR REPLACE FUNCTION
    user_management.get_following(p_user_id BIGINT)
RETURNS REFCURSOR AS $$
DECLARE
    following_cursor REFCURSOR;
BEGIN
    OPEN following_cursor FOR
    SELECT u.id, u.login, u.username, u.email, u.about, u.profile_image
    FROM user_management.follows f
    JOIN user_management."User" u ON f.following = u.id
    WHERE f.followers = p_user_id;

    RETURN following_cursor;
END;
$$ LANGUAGE plpgsql;
```

2. **Kursor: user_management.get_all_users**

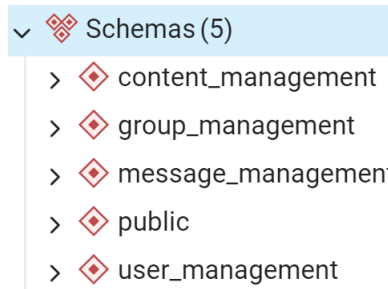
```
CREATE OR REPLACE FUNCTION user_management.get_all_users()
RETURNS REFCURSOR AS $$
DECLARE
    user_cursor REFCURSOR;
BEGIN

    OPEN user_cursor FOR
    SELECT u.id, u.login, u.username, u.email, u.password, u.about,
    u.profile_image
    FROM user_management."User" u;

    RETURN user_cursor;
END;
$$ LANGUAGE plpgsql;
```

Baza danych

Podział na schematy/pakiety



Rysunek 2. Pakiety Bazy Danych

Wdrożone mechanizmy

- **Integracja z aplikacją przez Web-Strone:** System umożliwia zarządzania bazą danych za pomocą interfejsu użytkownika.
- **Ograniczenie integralności:** Klucze obce i reguły integralności zapewniają spójność danych między tabelami.

Koncepcja dostępu zdalnego do bazy danych

Założenia

- Dostęp do bazy danych odbywa się za pośrednictwem Web-Strony, która łączy się z bazą danych za pomocą zdalnego połączenia.
- Aplikacja jest prosta w obsłudze i intuicyjna co zapewnia szybkie wdrożenie nowego użytkownika.
- Użytkownik może przeglądać, dodawać, edytować i usuwać dane.

Planowanie funkcjonalności

- Interfejs klienta: stworzenie interfejsu klienta, w którym będzie On mógł sprawdzać swoje informacje.
- Generowanie backup danych: Możliwość tworzenia kopii zapasowych bazy danych,co zapewni ochronę przed utratą danych

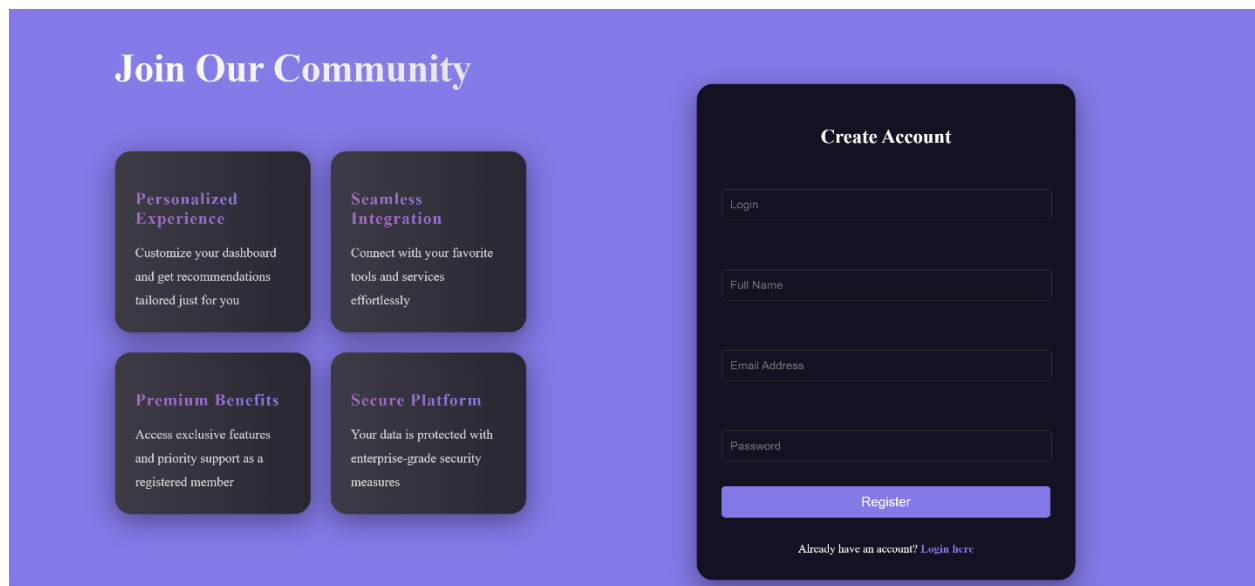
Realizacja dostępu zdalnego

Opis

W celu umożliwienia zdalnego dostępu stworzono aplikację webową opartą na React.js (frontend) oraz Spring (backend).

Prezentacja

Na rysunku 3 przedstawiono okno pojawiające się po wejściu na stronę do rejestracji użytkownika.



The image shows a registration window with a purple background. On the left, under the heading "Join Our Community", there are four dark gray boxes with white text:

- Personalized Experience**: Customize your dashboard and get recommendations tailored just for you.
- Seamless Integration**: Connect with your favorite tools and services effortlessly.
- Premium Benefits**: Access exclusive features and priority support as a registered member.
- Secure Platform**: Your data is protected with enterprise-grade security measures.

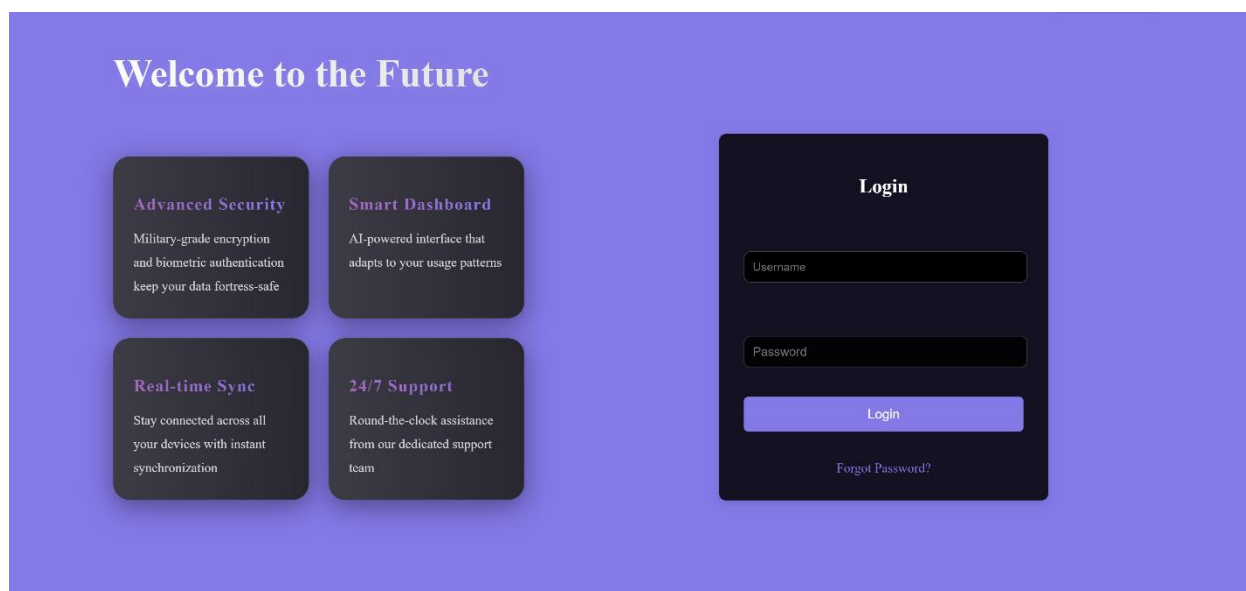
On the right, there is a dark gray "Create Account" form with the following fields:

- Login
- Full Name
- Email Address
- Password

Below the fields is a blue "Register" button. At the bottom of the form, it says "Already have an account? [Login here](#)".

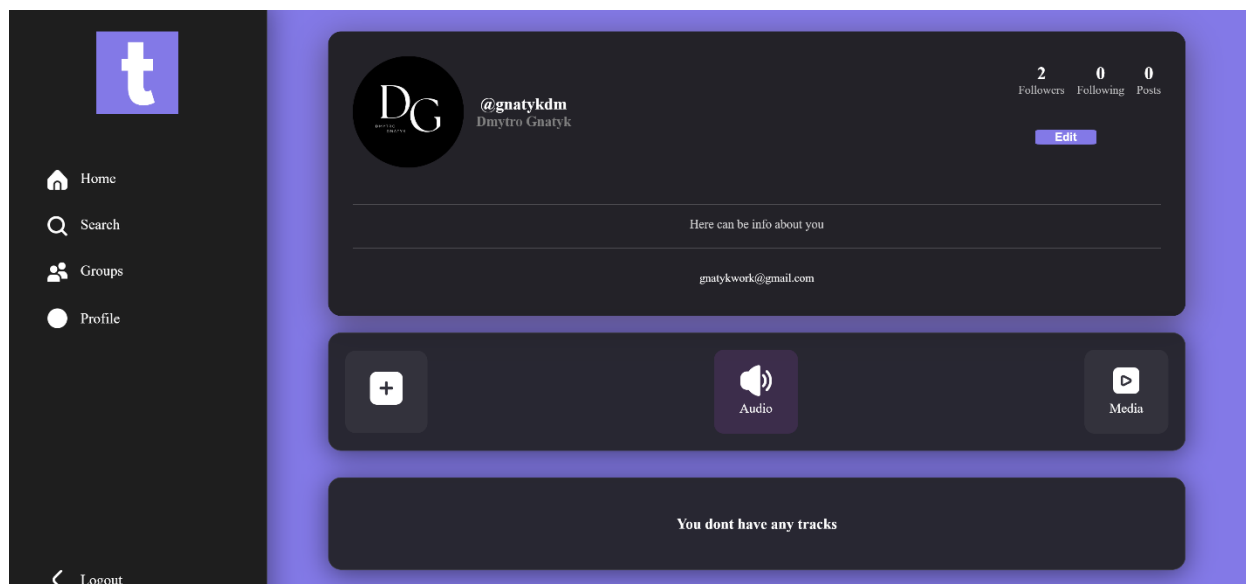
Rysunek 3. Okno Rejestracja

Na rysunku 4. Przedstawiono okno do logowania użytkownika do systemu.



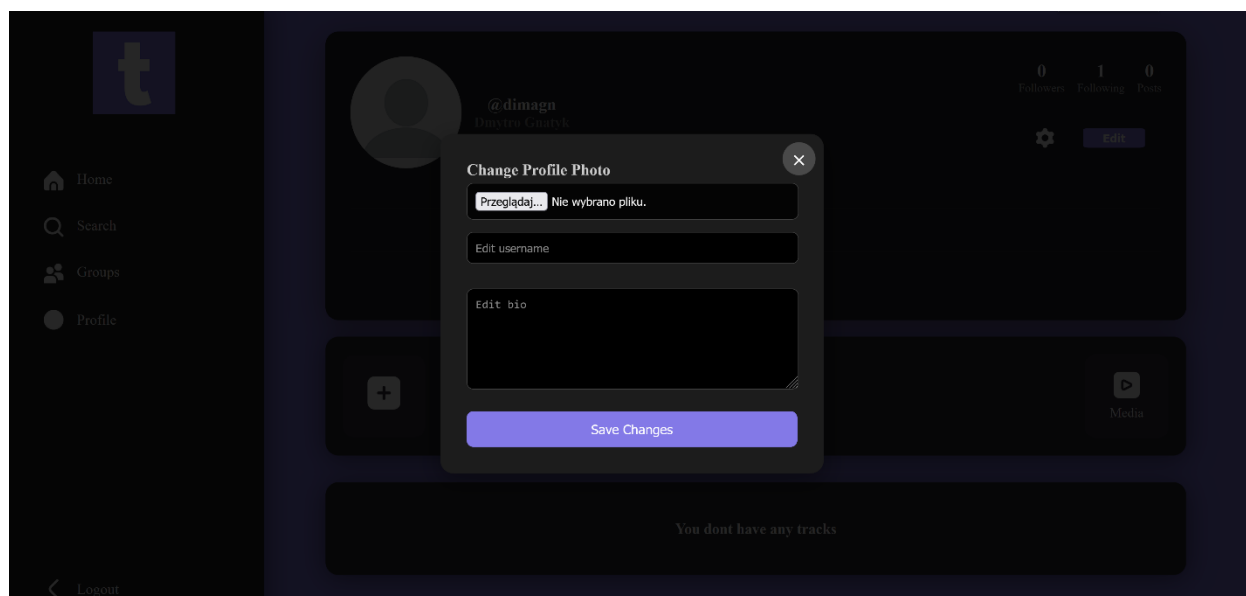
Rysunek 4. Okno logowania

Na rysunku 5. Przedstawiono profil użytkownika po udanej rejestracji lub procesie logowania się do systemu. Na którym są widoczne: zdjęcie użytkownika, login, imię, opis, liczba obserwujących i obserwowanych przez użytkownika ludzi.



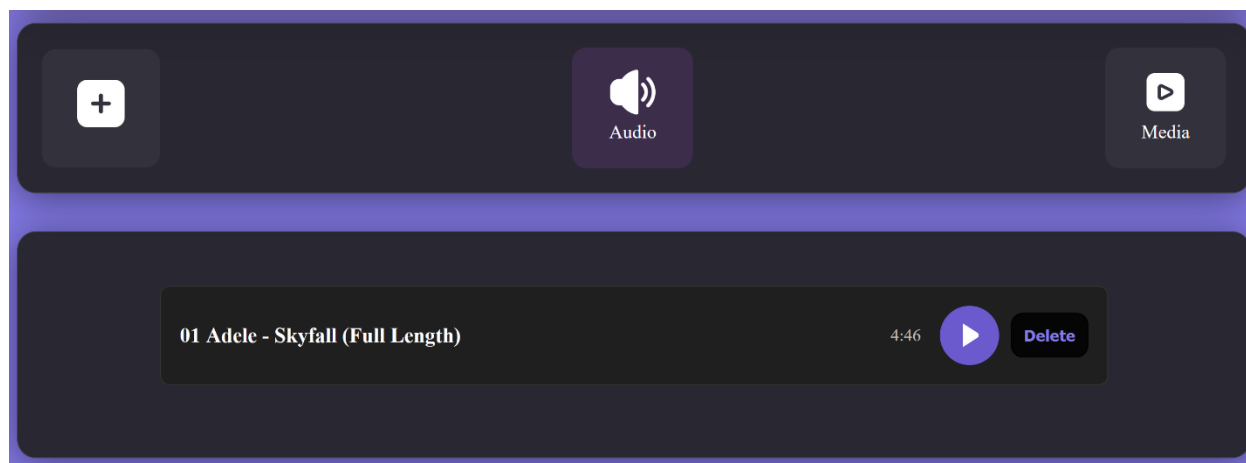
Rysunek 5. Okno Profilu Użytkownika

Na rysunku 6. Przedstawiono okno Edytowania danych użytkownika. W których można zmienić: zdjęcie profilowe, imię i opis użytkownika.



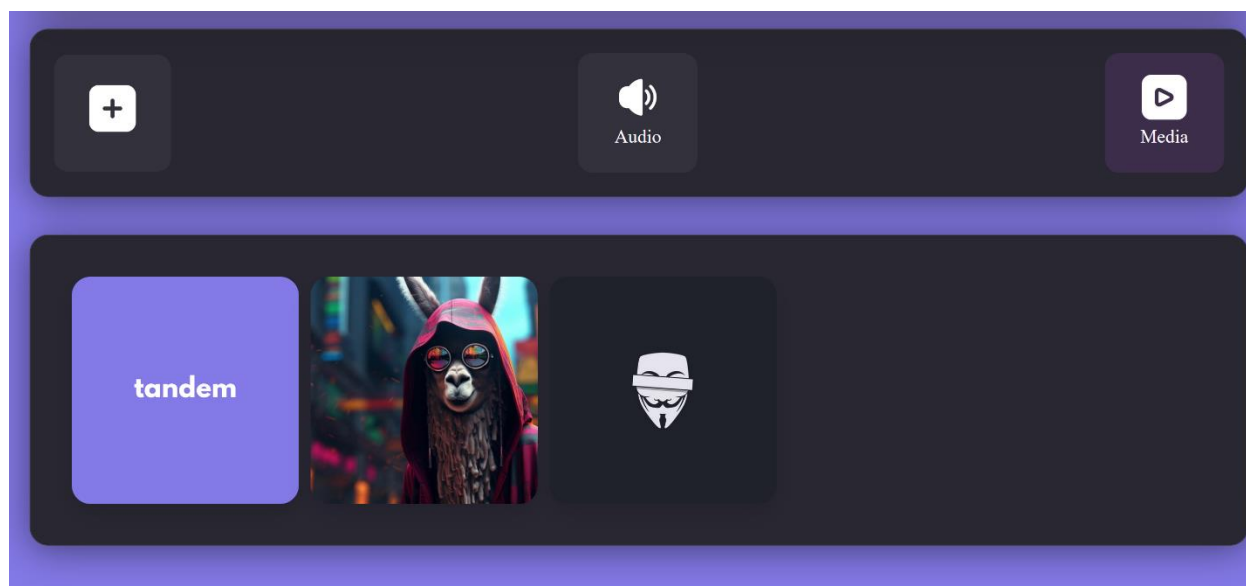
Rysunek 6. Okno edycji danych użytkownika

Na rysunku 7. Przedstawiono dane audio które można: usuwać i dodawać..



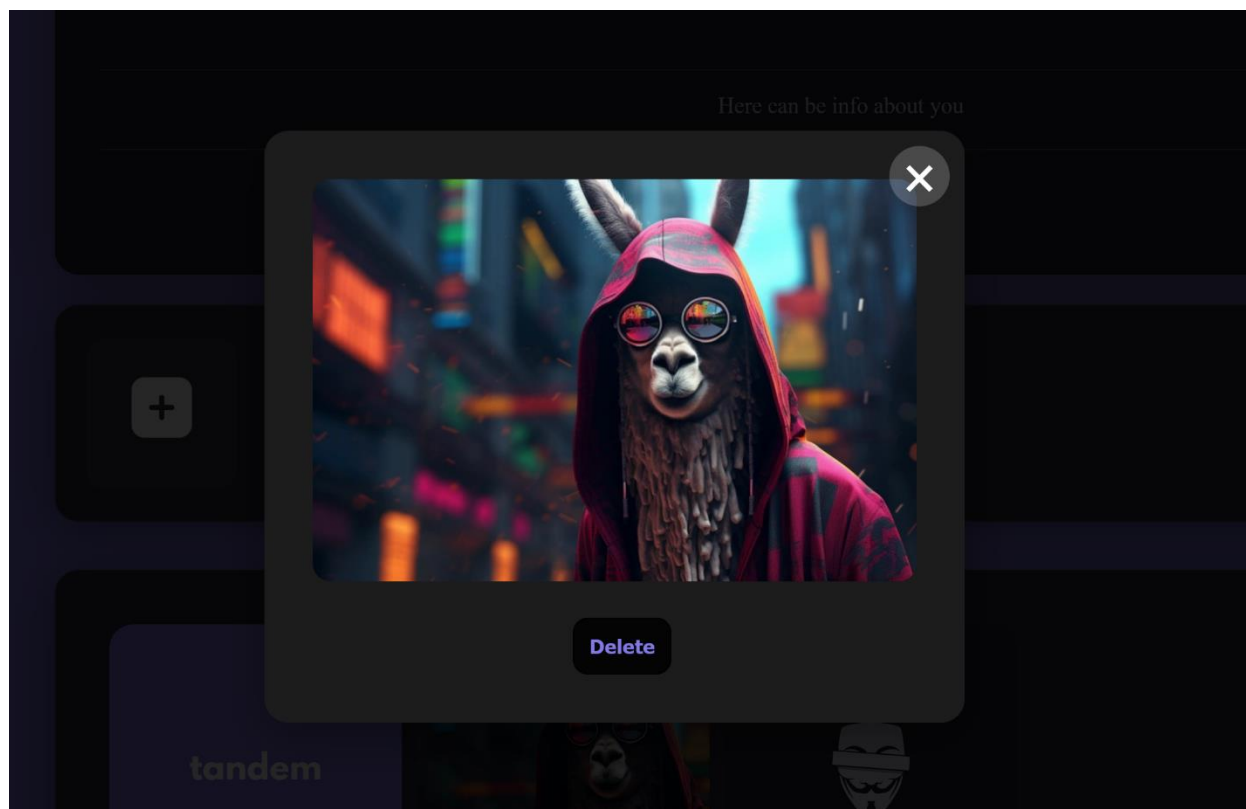
Rysunek 7. Okno audio użytkownika.

Na rysunku 8. Przedstawiono dane video i photo użytkownika które można dodawać.



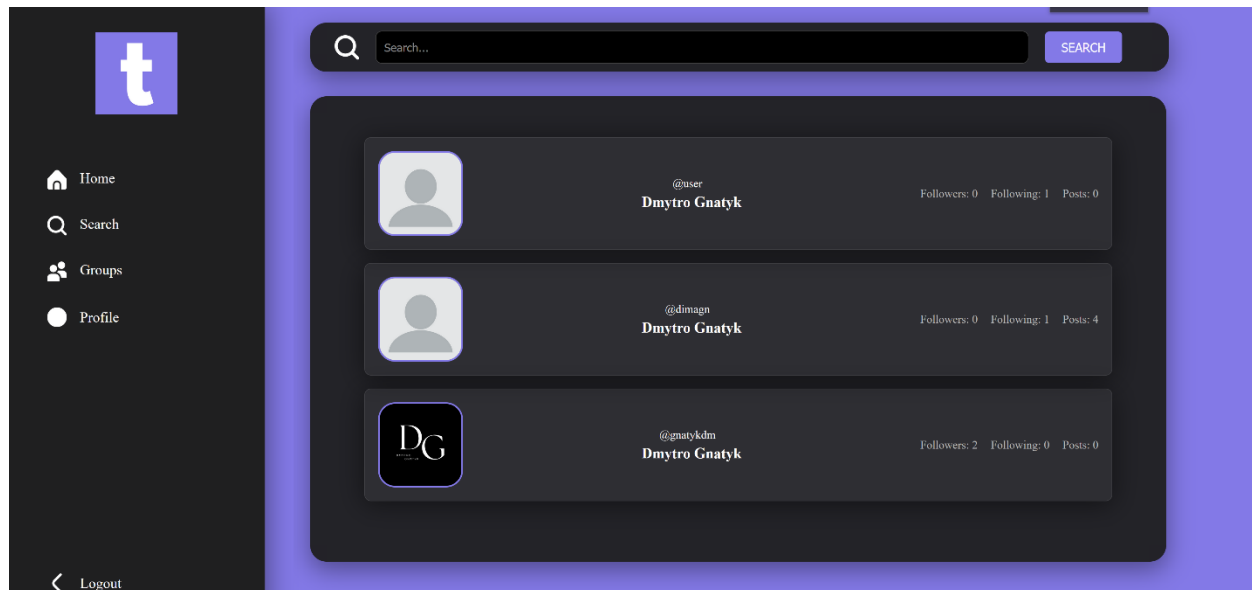
Rysunek 8. Okno video i zdjęcie użytkownika.

Na rysunku 9. Przedstawiono okno otwarte zdjęcie w który jest możliwość jego usunięcia.



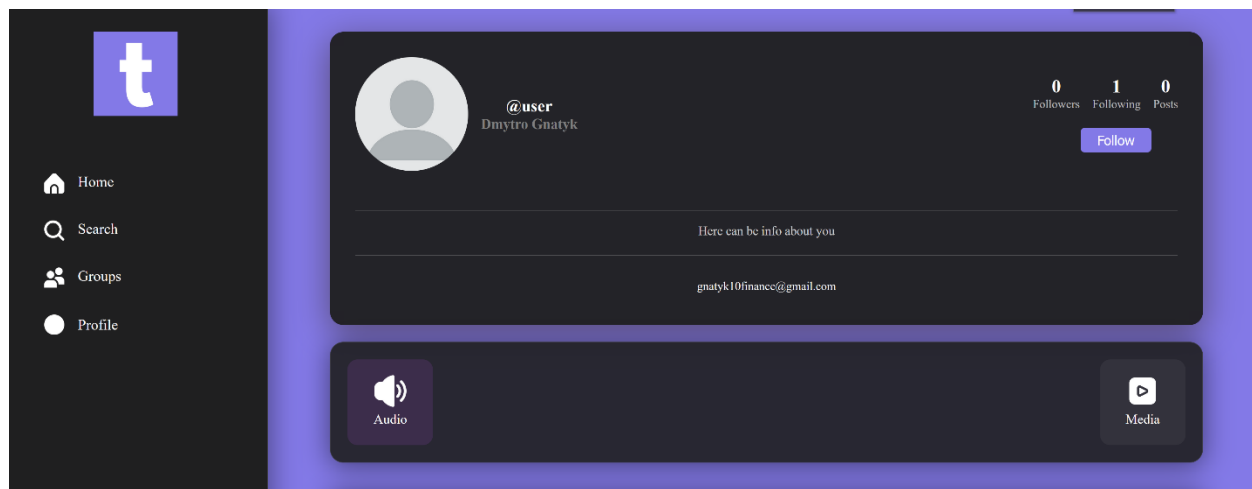
Rysunek 9. Okno zdjęcia

Na rysunku 10. Przedstawiono okno dla wyszukiwania innych użytkowników.



Rysunek 10. Okno wyszukiwania innych użytkowników.

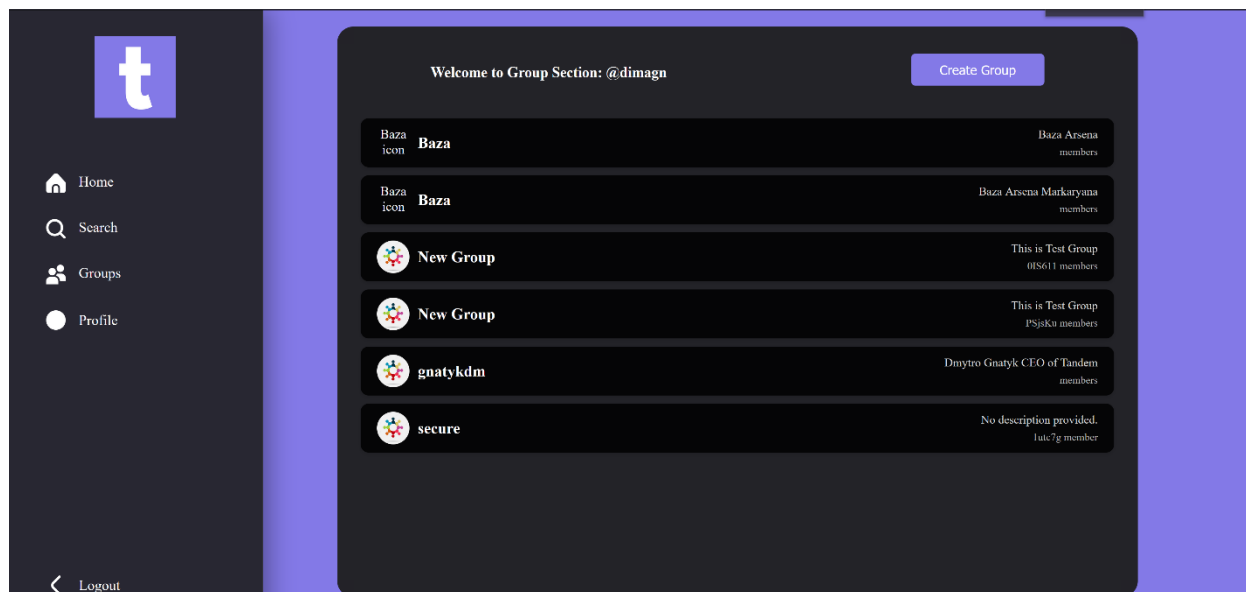
Na rysunku 9. Przedstawiono okno innego użytkownika którego można zaobserwować lub nie.



Rysunek 11. Okno innego użytkownika

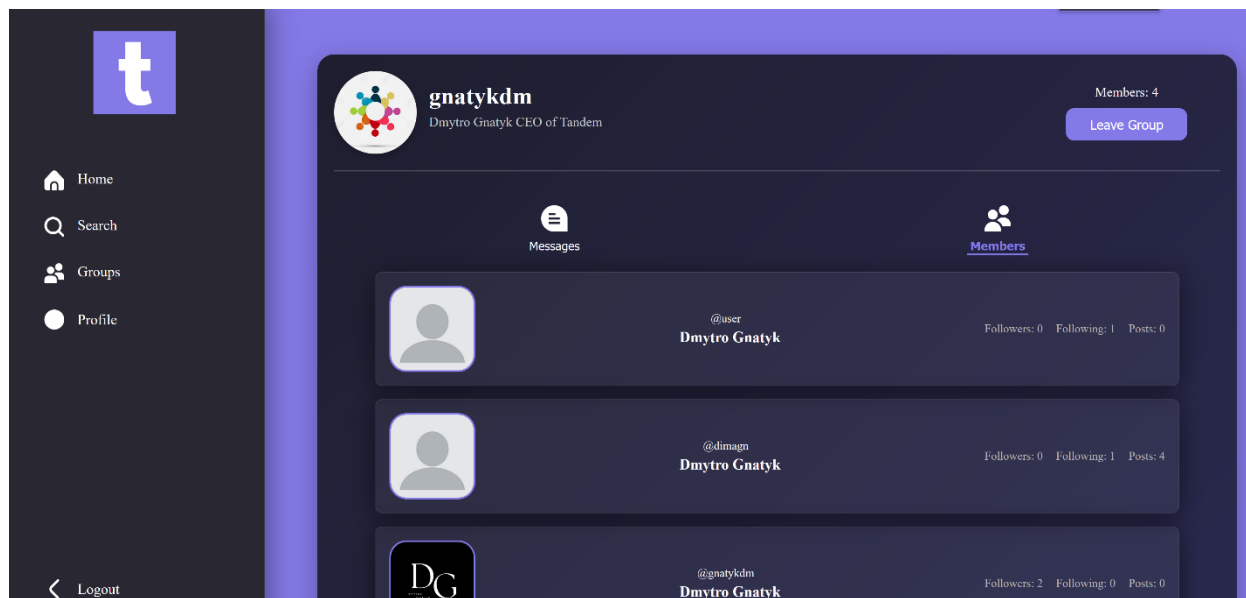
Na rysunku 10. Przedstawiono okno grup. Na którym widoczne pojedyncze grupy i

jest zawarty przycisk „Create Group” dla utworzenia nowej grupy.



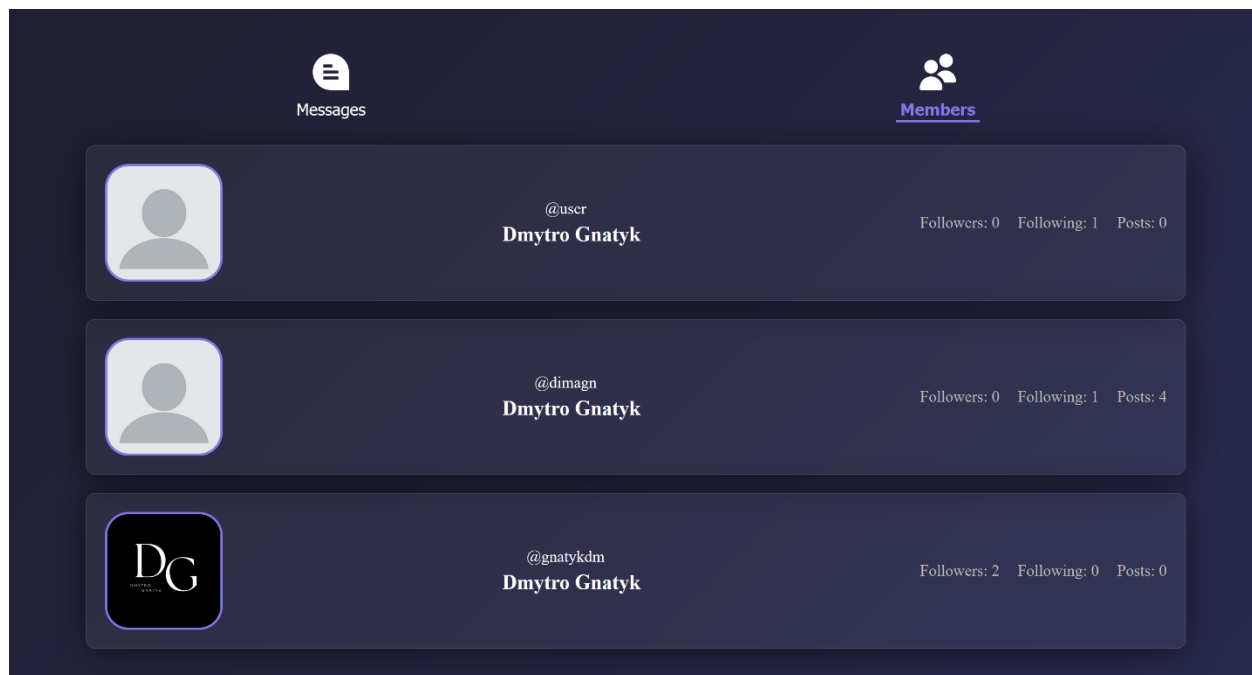
Rysunek 12. Okno dla wyszukiwania grup

Na rysunku 11. Przedstawiono okno profilu grupy na której pojawia się przycisk: Join, Leave oraz Delete jeżeli dany użytkownik jest administratorem.



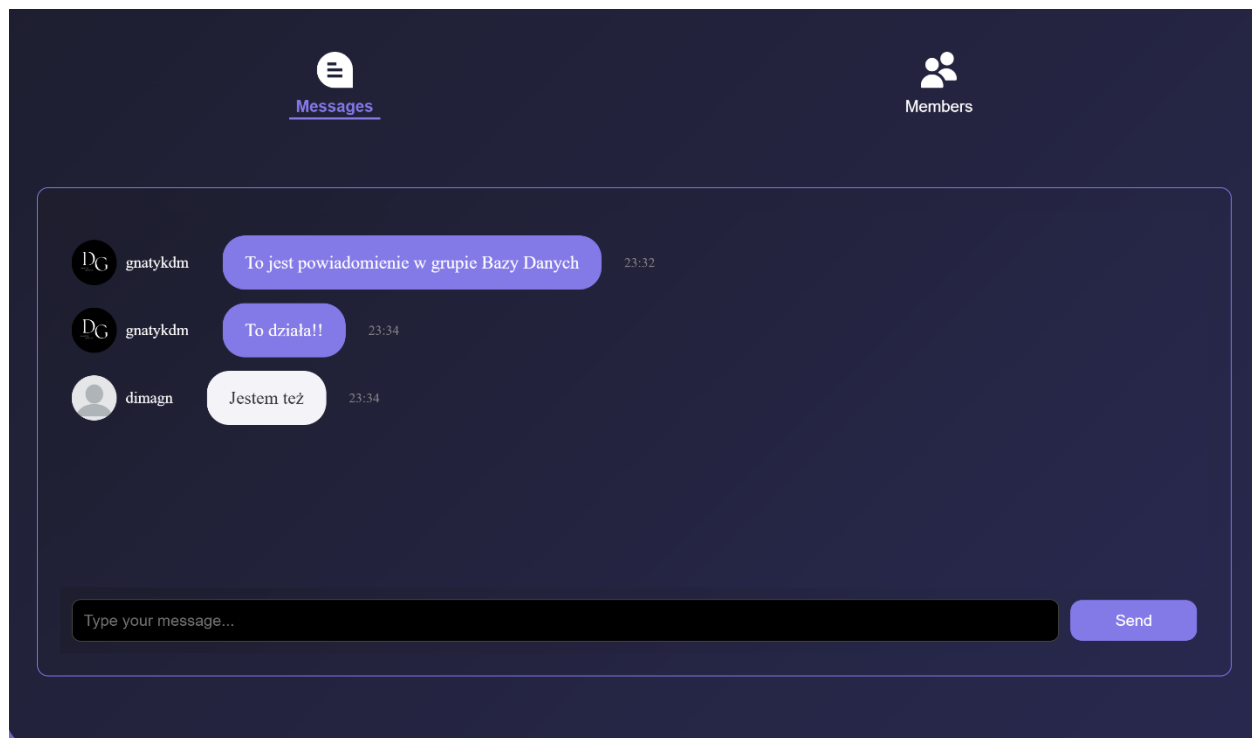
Rysunek 11. Okno profilowe grupy

Na rysunku 13. Przedstawiono okno użytkowników które znajdują w danej grupie.



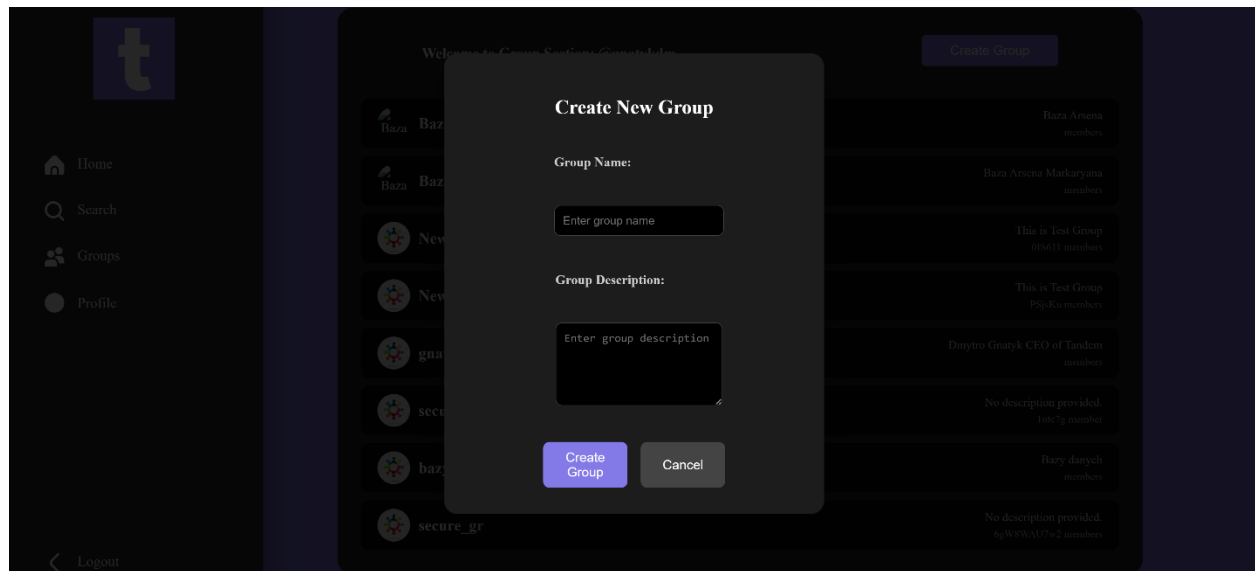
Rysunek 12. Użytkownicy które zawarte w grupie.

Na rysunku 13. Przedstawiono okno powiadomień grupy na który są zawarte: Zdjęcie użytkownika który wysłał powiadomienie, imię, zawartość i czas. Także jest pole dla dodawaniu tekstu.



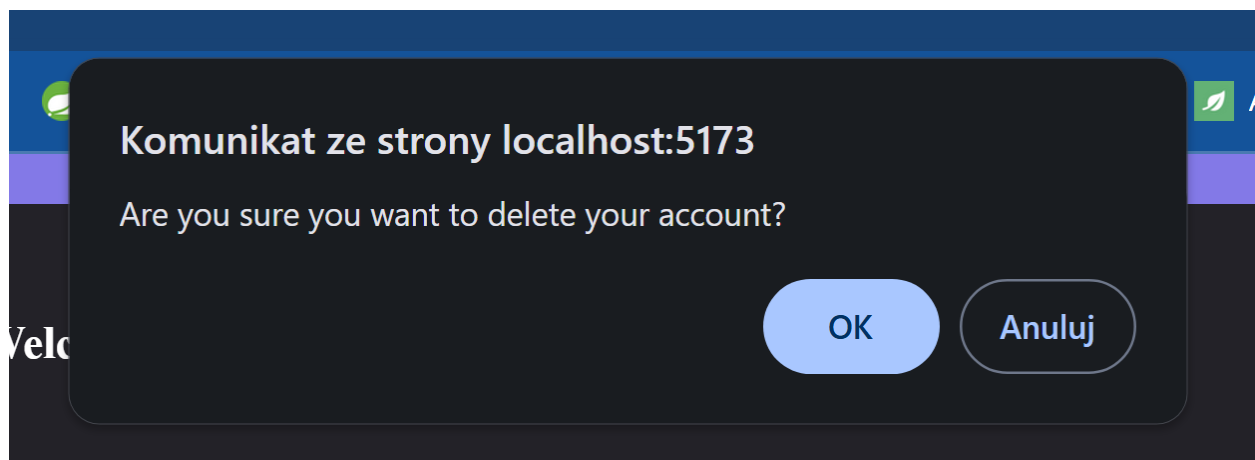
Rysunek 13. Okno powiadomień grupy.

Na rysunku 14. Przedstawiono okno dla tworzenia grupy.



Rysunek 15. Okno tworzenia grupy.

Na rysunku 15. Prezdstawiono okno dla usunięcia konta użytkownika.



Rysunek 15. Okno usunięcia konta użytkownika.

Opis Migracji bazy na model nierelacyjny

Migracja polega na przekształceniu relacyjnego modelu danych, który składa się z wielu połączonych tabel, w model nierelacyjny oparty na dokumentach. Proces ten upraszcza strukturę bazy danych, pozwalając na szybszy dostęp do danych i łatwiejsze skalowanie.

Struktura relacyjnej bazy danych i model docelowy

Relacyjna baza danych składa się z następujących tabel:

1. **Użytkownicy (user_management."User")**

- Przechowuje informacje o użytkownikach, takie jak: ID, login, nazwa

użytkownika, e-mail, hasło, opis i zdjęcie profilowe.

2. **Obserwacje (user_management.follows)**

- Reprezentuje relacje "obserwujący-obserwowany" między użytkownikami.

3. **Grupy (group_management."Group")**

- Zawiera informacje o grupach, takie jak: ID grupy, nazwa, ikona, opis, kod dostępu i typ grupy.

4. **Członkostwo w grupach (group_management.Optional)**

- Przechowuje informacje o członkach grup oraz ich statusie administratora.

5. **Wiadomości (message_management."Message")**

- Przechowuje wiadomości użytkowników, w tym nadawcę, treść i czas wysłania.

6. **Zawartość multimedialna:**

- **content_management.Photo:** Zdjęcia użytkowników.
- **content_management.Video:** Filmy użytkowników.
- **content_management.Audio:** Pliki audio użytkowników.
- **content_management.Content:** Ogólna tabela zawierająca odniesienia do zdjęć, filmów i audio.

Model docelowy (nierelacyjny)

Model nierelacyjny (MongoDB) opiera się na kolekcjach dokumentów JSON. Dane są grupowane w logiczne struktury z możliwością zagnieżdżenia i denormalizacji, co zmniejsza potrzebę wykonywania złożonych zapytań.

Przykładowe kolekcje w modelu docelowym:

1. **Users**

- Zawiera dane użytkowników, w tym listę obserwujących i obserwowanych.

2. **Groups**

- Przechowuje informacje o grupach wraz z ich członkami i administratorami.

3. **Messages**

- Przechowuje wiadomości użytkowników z informacjami o nadawcy i czasie wysłania.

4. **Content**

- Kolekcja zawiera dane multimedialne, takie jak zdjęcia, filmy i pliki audio.
-

Transformacja danych

Mapping tabel na kolekcje

Proces odwzorowania tabel relacyjnych na kolekcje JSON wygląda następująco:

Tabela relacyjna	Kolekcja JSON
user_management."User"	Users
user_management.follows	Embedded w Users
group_management."Group"	Groups
group_management.Optional	Embedded w Groups
message_management."Message"	Messages
content_management.Photo	Embedded w Content
content_management.Video	Embedded w Content
content_management.Audio	Embedded w Content

Zagnieżdżenie danych

Zagnieżdżenie danych pozwala na umieszczanie powiązanych informacji w jednym dokumencie. Przykłady:

- Informacje o obserwujących są osadzone bezpośrednio w kolekcji Users.
- Dane o członkach i administratorach grup są osadzone w dokumentach Groups.

Przykład dokumentu w kolekcji Users:

```
{
  "id": 1,
  "username": "JanKowalski",
  "email": "jan.kowalski@example.com",
  "profile_image": "image_url",
  "about": "Opis użytkownika",
  "followers": [
    {"id": 2, "username": "AnnaNowak"},
    {"id": 3, "username": "PiotrZielinski"}
  ],
  "following": [
    {"id": 4, "username": "KasiaKowalska"}
  ]
}
```

Denormalizacja danych

W celu uproszczenia zapytań stosujemy denormalizację, czyli powielanie danych. Przykład: W kolekcji Groups dane o użytkownikach są powielane, aby szybciej uzyskać dostęp do informacji o członkach.

Przykład dokumentu w kolekcji Groups:

```
{
  "group_id": 1,
  "group_name": "Grupa Programistów",
  "group_icon": "icon_url",
  "group_description": "Grupa dla miłośników programowania",
  "members": [
    {"id": 1, "username": "JanKowalski", "admin": true},
    {"id": 2, "username": "AnnaNowak", "admin": false}
  ]
}
```

Przykłady wstawiania danych

Wstawianie danych w MongoDB

Wstawianie pojedynczego użytkownika:

```
db.Users.insertOne({
  id: 1,
  username: "JanKowalski",
  email: "jan.kowalski@example.com",
  profile_image: "image_url",
  about: "Opis użytkownika",
  followers: [],
  following: []
});
```

Wstawianie wielu dokumentów do kolekcji platformy

```
db.Groups.insertMany([
  {
    group_id: 1,
    group_name: "Grupa Programistów",
    group_icon: "icon_url",
    group_description: "Grupa dla miłośników programowania",
    members: [
      {"id": 1, "username": "JanKowalski", "admin": true},
      {"id": 2, "username": "AnnaNowak", "admin": false}
    ]
  },
  {
    group_id: 2,
    group_name: "Grupa Muzyczna",
    group_icon: "music_icon_url",
    group_description: "Grupa dla fanów muzyki",
    members: [
      {"id": 3, "username": "PiotrZielinski", "admin": true}
    ]
  }
]);
```

Porównanie SQL i MongoDB

SQL (Relacyjna)	MongoDB (Nierelacyjna)
Tabele z powiązaniem	Kolekcje JSON z osadzonymi dokumentami
Normalizacja danych	Denormalizacja danych
Złożone zapytania JOIN	Prostsze, szybkie zapytania

Funkcje asynchroniczne dla MongoDB

MongoDB wspiera funkcje asynchroniczne, np. w Node.js:

```
async function addUser(user) {
  await db.Users.insertOne(user);
  console.log("Użytkownik dodany!");
}
```

Operacje CRUD w MongoDB

- **Create:** Wstawianie danych:
`db.Users.insertOne({username: "JanKowalski", email: "jan.kowalski@example.com"});`
- **Read:** Pobieranie danych:
`db.Users.findOne({username: "JanKowalski"});`
- **Update:** Aktualizacja danych:
`db.Users.updateOne({username: "JanKowalski"}, {$set: {email: "nowy_email@example.com"}});`
- **Delete:** Usuwanie danych:
`db.Users.deleteOne({username: "JanKowalski"});`

Dzięki zastosowaniu modelu nierelacyjnego baza danych staje się bardziej elastyczna i łatwiejsza do zarządzania, zwłaszcza w przypadku aplikacji o dużej ilości danych powiązanych.