DataBox

System przechowywania notatek i obrazów

Dokumentacja techniczna

Projekt Semestralny
Bazy danych

Prowadzący: mgr inż. Aleksander Wojtowicz

Autor: Przemysław Sulowski

Nr indeksu: 134975

Next.js | .NET | TypeScript | PostgreSQL

Rzeszów, 17 stycznia 2026

Uniwersytet Rzeszowski

Streszczenie

Niniejsza dokumentacja przedstawia aplikację webową do przechowywania i organizowania zdjęć, notatek oraz linków, która umożliwia użytkownikom bezpieczne zapisywanie treści, porządkowanie ich w kategoriach, szybkie wyszukiwanie oraz łatwy dostęp do zgromadzonych materiałów z dowolnego urządzenia w czasie rzeczywistym.

Projekt został zrealizowany w architekturze klient-serwer z wykorzystaniem technologii .NET 10 na backendzie oraz Next.js 16 z React 19 i TypeScript na frontendzie. Aplikacja korzysta z bazy danych PostgreSQL 17.

Słowa kluczowe: przechowywanie, obrazy, notatki, linki, React, .NET 8, TypeScript, PostgreSQL.

Spis treści

Spis rysunków

Spis tabel

Listings

1 Wprowadzenie

1.1. Kontekst i motywacja

Zarządzanie i organizacja informacji cyfrowych stanowią kluczowy element funkcjonowania współczesnego człowieka. W erze dynamicznego rozwoju technologii tradycyjne metody przechowywania danych, takie jak lokalne pliki czy rozproszone notatki, stają się niewystarczające. Istnieje rosnące zapotrzebowanie na nowoczesne aplikacje webowe, które umożliwiają centralne gromadzenie zdjęć, notatek oraz linków, ich porządkowanie w kategoriach oraz szybki dostęp do zgromadzonych treści z dowolnego urządzenia.

1.2. Cel projektu

Głównym celem projektu było stworzenie nowoczesnej, bezpiecznej i wydajnej aplikacji webowej klasy produkcyjnej, umożliwiającej całościowe przechowywanie i organizowanie zdjęć, notatek oraz linków. Aplikacja ma być intuicyjna dla użytkownika końcowego, a jednocześnie zaawansowana technologicznie „pod maską".

1.3. Zakres projektu

Projekt obejmuje pełen cykl wytwarzania oprogramowania, od analizy wymagań po wdrożenie na serwerze produkcyjnym.

- **Backend:** REST API w .NET 10 [1].
- **Frontend:** Next.js 16 [2] z React 19 [3] i TypeScript [4].
- **Baza danych:** Relacyjna baza PostgreSQL 17 [5].

1.4. Użyte technologie

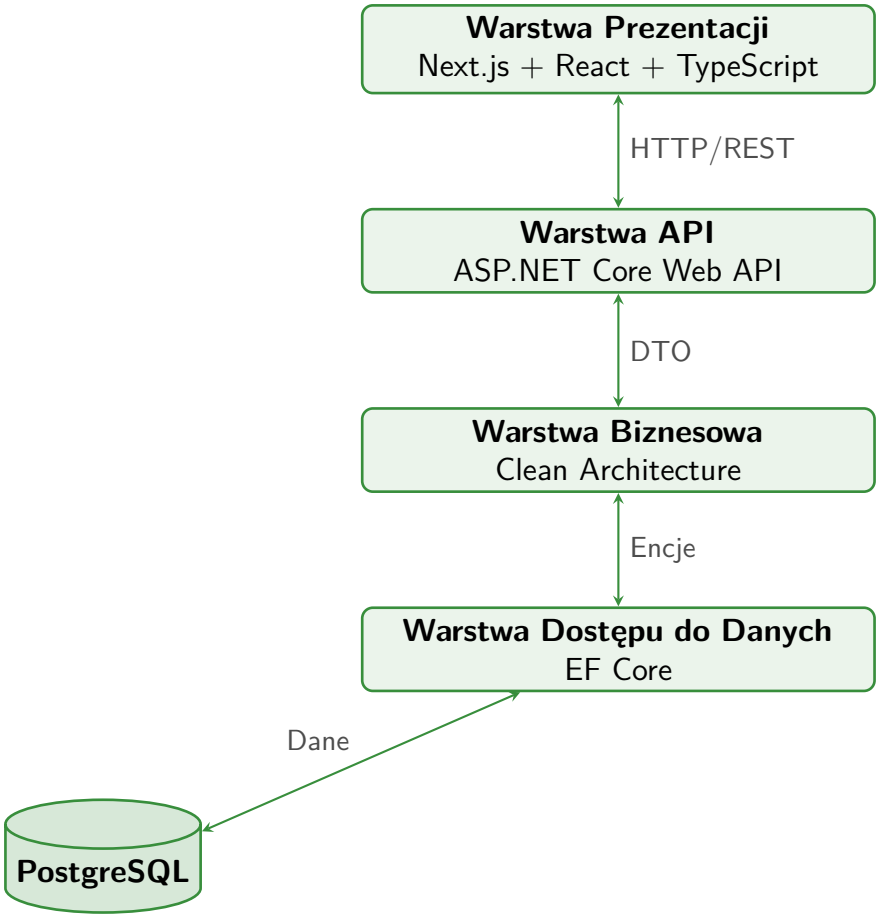
Tabela 1: Stack technologiczny projektu

Kategoria	Technologia	Szczegóły / Uzasadnienie
Backend & Dane		
Platforma	**.NET 10.0**	Najnowsza wersja
ORM	**EF Core 8.0**	Podejście Code First, wysoka wydajność
Baza danych	**PostgreSQL 17**	Zaawansowany, obiektowo-relacyjny silnik SQL
Frontend & UI		
Framework	**Next.js 16.1**	Framework oparty na React, umożliwiający pełne zarządzanie aplikacją webową
Framework	**React 19.2**	Najpopularniejsza biblioteka komponentowa
Język	**TypeScript 5.9**	Statyczne typowanie zapewniające bezpieczeństwo

2 Architektura systemu

2.1. Architektura wysokopoziomowa

System został zaprojektowany w architekturze trójwarstwowej z wyraźnym podziałem odpowiedzialności, co ułatwia jego rozwój, testowanie i utrzymanie [6]. Komunikacja między klientem a serwerem odbywa się bezstanowo poprzez protokół HTTPS.



Rysunek 1: Architektura wysokopoziomowa systemu

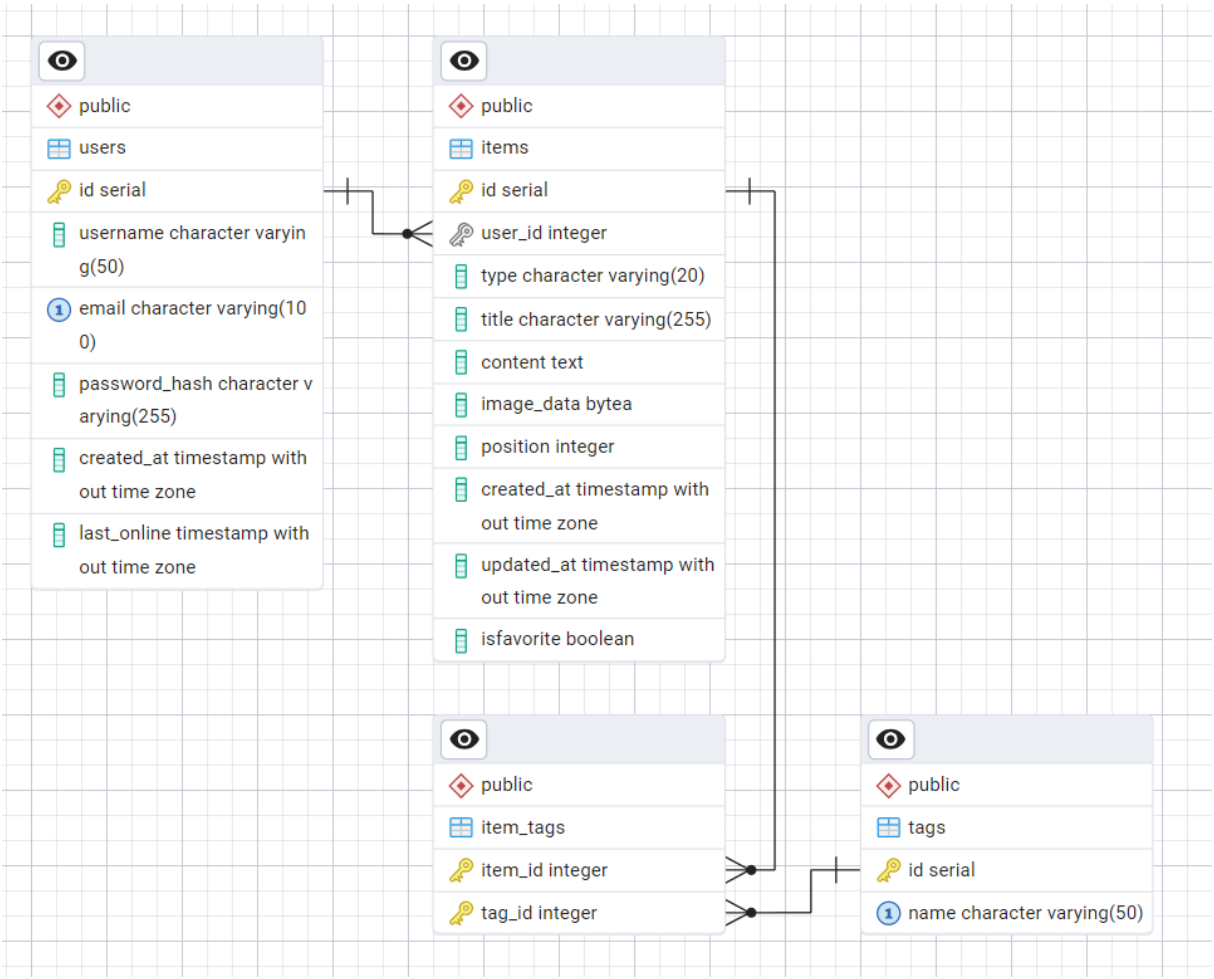
2.2. Wzorce architektoniczne

Projekt backendu opiera się na zasadach **Clean Architecture** zaproponowanych przez Roberta C. Martina [6], co zapewnia separację logiki biznesowej od szczegółów implementacyjnych (takich jak baza danych czy framework UI). Dodatkowo w projekcie zastosowano następujące wzorce projektowe:

- **Repository Pattern:** Abstrakcja warstwy dostępu do danych, pozwalająca na uniezależnienie logiki biznesowej od konkretnego silnika bazy danych [7].

3 Model danych

3.1. Diagram ERD



Rysunek 2: Diagram ERD

3.2. Opis tabel

- **users** — tabela przechowująca dane użytkowników:
  - `id SERIAL PRIMARY KEY` — klucz główny
  - `username VARCHAR(50)` — nazwa użytkownika
  - `email VARCHAR(100)` — adres e-mail
  - `password_hash VARCHAR(255)` — zaszyfrowane hasło
  - `created_at TIMESTAMP` — data utworzenia konta
  - `last_online TIMESTAMP` — ostatnia aktywność użytkownika

- **items** — tabela przechowująca elementy użytkownika (zdjęcia, notatki, linki):
  - `id SERIAL PRIMARY KEY`
  - `user_id INTEGER REFERENCES users(id)` — klucz obcy do tabeli `users`
  - `type VARCHAR(20)` — typ elementu (zdjęcie, notatka, link)
  - `title VARCHAR(255)` — tytuł elementu
  - `content TEXT` — zawartość elementu
  - `image_data BYTEA` — dane obrazu (jeśli typ = zdjęcie)
  - `position INTEGER` — pozycja elementu w liście
  - `created_at TIMESTAMP` — data utworzenia
  - `updated_at TIMESTAMP` — data modyfikacji
  - `isfavorite BOOLEAN` — oznaczenie elementu jako ulubionego
- **tags** — tabela tagów:
  - `id SERIAL PRIMARY KEY`
  - `name VARCHAR(50)` — nazwa tagu
- **item_tags** — tabela łącząca elementy z tagami (relacja N:M):
  - `item_id INTEGER REFERENCES items(id)`
  - `tag_id INTEGER REFERENCES tags(id)`
  - **PRIMARY KEY (item_id, tag_id)**

3.3. Relacje między encjami

- **users** — **items** : 1:N (jeden użytkownik może mieć wiele elementów)
- **items** — **tags** : N:M (element może mieć wiele tagów, tag może być przypisany do wielu elementów), realizowane przez tabelę pośrednią `item_tags`

3.4. Klucze i indeksy

- Klucze główne: `users.id`, `items.id`, `tags.id`, `item_tags(item_id, tag_id)`
- Klucze obce: `items.user_id` → `users.id`, `item_tags.item_id` → `items.id`, `item_tags.tag_id` → `tags.id`
- Indeksy opcjonalne dla optymalizacji zapytań:

- indeks na `items.user_id` (szybkie wyszukiwanie elementów danego użytkownika)
- indeks na `tags.name` (wyszukiwanie tagów)

4 Implementacja kluczowych funkcjonalności

4.1. Zarządzanie użytkownikami

Zaimplementowana została kompleksowa obsługa użytkowników po stronie API, w którym zrealizowany jest w pełni CRUD (Create, Read, Update, Delete).

```
[HttpGet("{id}")]
public async Task<ActionResult<UserDto>> GetUser(int id)
{
    var user = await _context.Users
                             .Where(u => u.Id == id)
                             .ProjectTo<UserDto>(_mapper.
    ConfigurationProvider)
                             .FirstOrDefaultAsync();

    if (user == null)
        return NotFound();
    return Ok(user);
}
```

Listing 1: Pobieranie użytkownika po id

```
[HttpPost("login")]
public async Task<ActionResult<int>> Login(LoginDto loginDto)
{
    // funkcja hashująca
    var passwordHash = PasswordHasher.Hash(loginDto.Password);

    var user = await _context.Users
        .FirstOrDefaultAsync(u =>
            u.Email == loginDto.Email &&
            u.PasswordHash == passwordHash);
    if (user == null)
        return Unauthorized(new { message = "niepoprawny login lub hasło" });

    user.LastOnline = DateTime.Now;
    await _context.SaveChangesAsync();

    var result = new
    {
        id = user.Id,
```

```
        username = user.Username
    };

    return Ok(result);
}
```

Listing 2: Logowanie użytkownika

```
[HttpPost("register")]
public async Task<ActionResult<UserDto>> PostUser(UserCreateDto newUser)
{
    if (await _context.Users.AnyAsync(u => u.Email == newUser.Email))
        return Conflict("Email zajęty.");

    var user = _mapper.Map<User>(newUser);

    // funkcja hashująca
    user.PasswordHash = PasswordHasher.Hash(newUser.Password);

    user.Items = new List<Item>();

    _context.Users.Add(user);
    await _context.SaveChangesAsync();

    var userDto = _mapper.Map<UserDto>(user);
    return CreatedAtAction(nameof(GetUser), new { id = user.Id }, userDto);
}
```

Listing 3: Rejestracja użytkownika

4.2. Zarządzanie zasobami

```
[HttpGet("items/{userId}")]
public async Task<ActionResult<IEnumerable<ItemDto>>> GetItemsByUser(int
    userId)
{
    var items = await _context.Items
        .Where(i => i.UserId == userId)
        .Include(i => i.Tags)
        .OrderBy(i => i.Position)
```

```
        .ToListAsync();

    var itemDtos = _mapper.Map<List<ItemDto>>(items);

    return Ok(itemDtos);
}
```

Listing 4: Pobieranie zasobów użytkownika

```
[HttpPost("items/add")]
public async Task<ActionResult<ItemDto>> AddItem([FromForm] ItemCreateDto
    newItem)
{
    var user = await _context.Users
        .Include(u => u.Items)
        .FirstOrDefaultAsync(u => u.Id == newItem.UserId);
    if (user == null)
        return NotFound("Użytkownik nie istnieje.");

    var item = _mapper.Map<Item>(newItem);
    if (newItem.ImageFile != null)
    {
        using var ms = new MemoryStream();
        await newItem.ImageFile.CopyToAsync(ms);
        item.ImageData = ms.ToArray();
    }

    if (newItem.TagIds != null && newItem.TagIds.Any())
    {
        var tags = await _context.Tags
            .Where(t => newItem.TagIds.Contains(t.Id))
            .ToListAsync();
        item.Tags = tags;
    }

    var maxPosition = await _context.Items
        .Where(i => i.UserId == newItem.UserId)
        .MaxAsync(i => (int?)i.Position) ?? 0;
    item.Position = maxPosition + 1;

    _context.Items.Add(item);
```

```
32      await _context.SaveChangesAsync();
33
34      var itemDto = _mapper.Map<ItemDto>(item);
35
36      return Ok();
37  }
```

Listing 5: Dodawanie zasobu użytkownika

```
1   [HttpDelete("items/{userId}/{itemId}")]
2   public async Task<IActionResult> DeleteItem(int userId, int itemId)
3   {
4       // pobieramy użytkownika wraz z jego itemami
5       var user = await _context.Users
6           .Include(u => u.Items)
7           .FirstOrDefaultAsync(u => u.Id == userId);
8
9       if (user == null)
10          return NotFound("Nie znaleziono użytkownika");
11
12      // szukamy itemu w kolekcji użytkownika
13      var item = user.Items.FirstOrDefault(i => i.Id == itemId);
14      if (item == null)
15          return NotFound("Nie znaleziono itemu");
16
17      // usuwamy item
18      _context.Items.Remove(item);
19      await _context.SaveChangesAsync();
20
21      return NoContent();
22  }
```

Listing 6: Usuwanie zasobu użytkownika

```
1   [HttpPut("items/update")]
2   public async Task<ActionResult<ItemDto>> UpdateItem([FromForm] ItemUpdateDto
        updatedItem)
3   {
4       var item = await _context.Items
5           .Include(i => i.Tags)
6           .FirstOrDefaultAsync(i => i.Id == updatedItem.Id && i.UserId ==
```

```
    updatedItem.UserId);
    if (item == null)
        return NotFound("Item nie istnieje lub nie należy do użytkownika.");

    item.Title = updatedItem.Title ?? item.Title;
    item.Content = updatedItem.Content ?? item.Content;
    item.Type = updatedItem.Type ?? item.Type;

    if (updatedItem.ImageFile != null)
    {
        using var ms = new MemoryStream();
        await updatedItem.ImageFile.CopyToAsync(ms);
        item.ImageData = ms.ToArray();
    }

    if (updatedItem.TagIds != null)
    {
        var tags = await _context.Tags
            .Where(t => updatedItem.TagIds.Contains(t.Id))
            .ToListAsync();
        item.Tags.Clear();
        (item.Tags as List<Tag>)?.AddRange(tags);
    }

    await _context.SaveChangesAsync();
    var itemDto = _mapper.Map<ItemDto>(item);

    return Ok(itemDto);
}
```

Listing 7: Aktualizacja zasobu użytkownika

```
[HttpPut("items/{userId}/positions")]
public async Task<IActionResult> UpdateItemPositions(
    int userId,
    [FromBody] List<ItemOrderDto> order)
{
    if (order == null || order.Count == 0)
        return BadRequest("Brak danych");

    var itemIds = order.Select(o => o.Id).ToList();
```

```
10
11      var items = await _context.Items
12          .Where(i => i.UserId == userId && itemIds.Contains(i.Id))
13          .ToListAsync();
14
15      if (items.Count != order.Count)
16          return BadRequest("Nieprawidłowe ID item'ow");
17
18      foreach (var item in items)
19      {
20          var newPosition = order.First(o => o.Id == item.Id).Position;
21          item.Position = newPosition;
22      }
23
24      await _context.SaveChangesAsync();
25
26      return NoContent();
27  }
```

Listing 8: Aktualizacja kolejności zasobów użytkownika

5 Frontend / Interfejs użytkownika

5.1. Komponenty UI (Next.js & React-Bootstrap)

Do zbudowania przejrzystego interfejsu użytkownika wykorzystano framework Next.js wraz z biblioteką React-Bootstrap. Dzięki temu możliwe było tworzenie nowoczesnych komponentów, takich jak okna modalne, przyciski czy formularze, przy jednoczesnym zachowaniu spójnego wyglądu aplikacji.

5.2. Customowe ikonki (React-Icons)

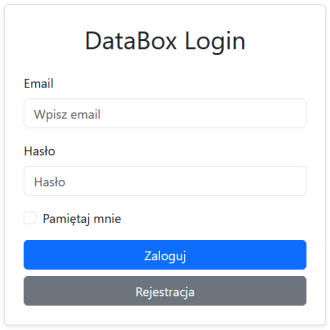
W celu zwiększenia czytelności interfejsu zastosowano niestandardowe ikonki z biblioteki `React-Icons`. Ikonki pomagają w szybkim rozpoznawaniu funkcji oraz poprawiają ogólne doświadczenie użytkownika.

5.3. Responsywność

Aplikacja jest w pełni responsywna. Użytkownik może korzystać z niej zarówno na komputerach stacjonarnych, jak i na urządzeniach mobilnych. Dzięki temu narzędzie pracy jest zawsze dostępne i wygodne w użyciu.

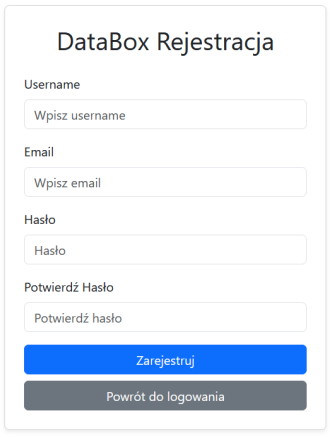
6 Zrzuty ekranu

6.1. Ekran logowania



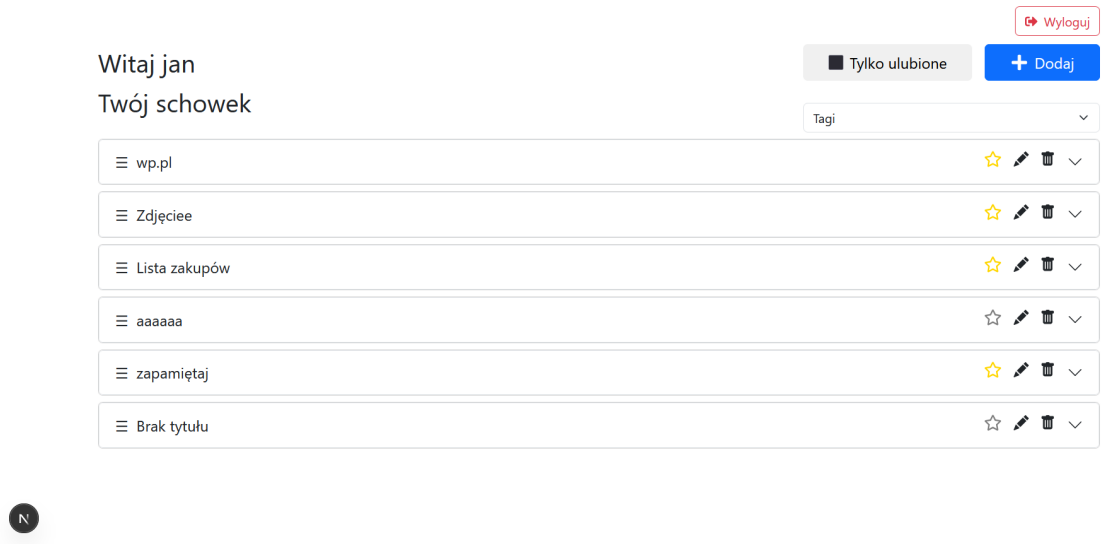
Rysunek 3: Formularz logowania

6.2. Ekran rejestracji



Rysunek 4: Formularz rejestracji

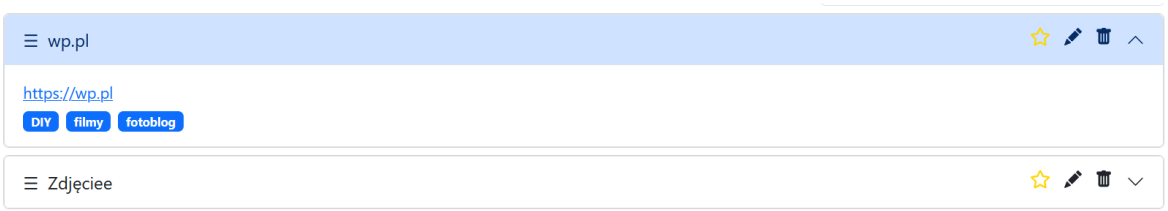
6.3. Dashboard



Rysunek 5: Ekran główny aplikacji

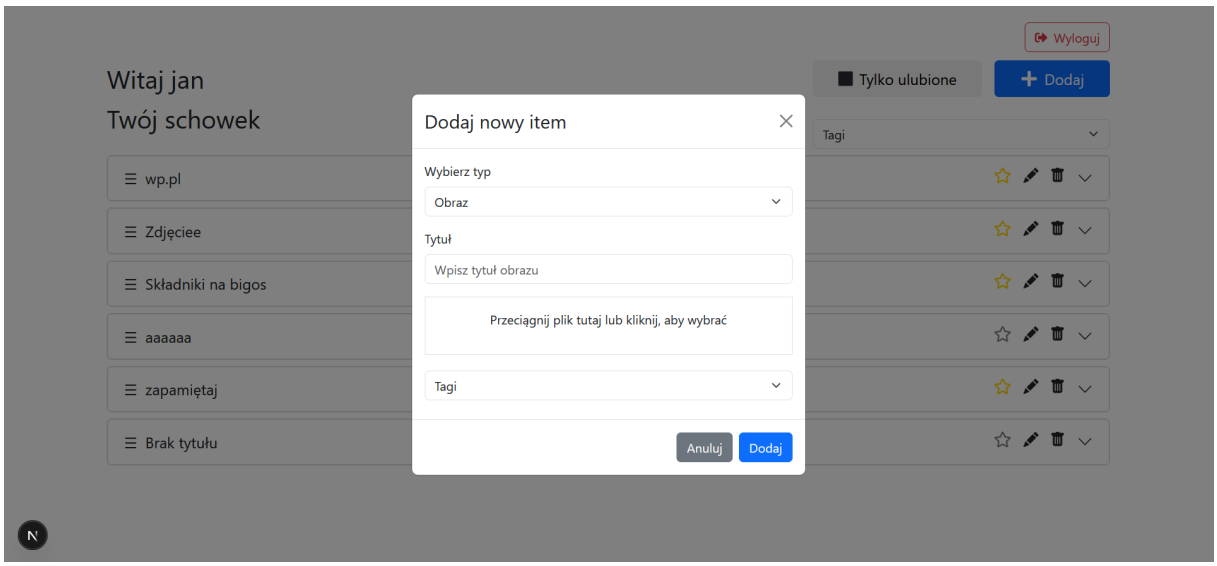
6.4. Zarządzanie zasobami

6.4.1. Rozwinięty widok zasobu



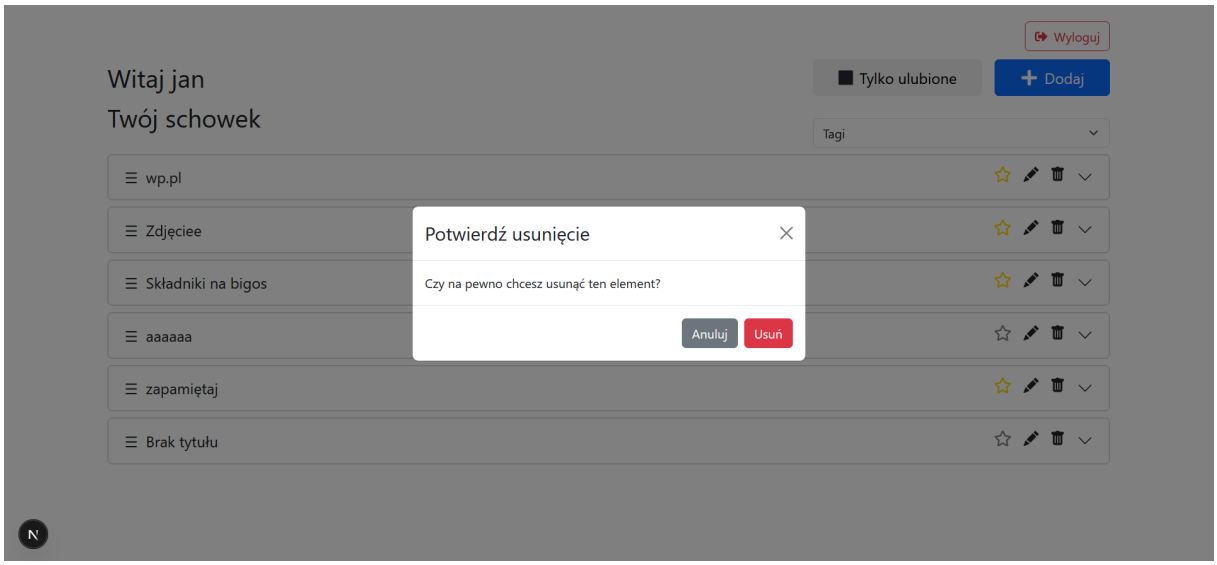
Rysunek 6: Rozwinięty zasobób użytkownika

6.4.2. Dodawanie zasobu



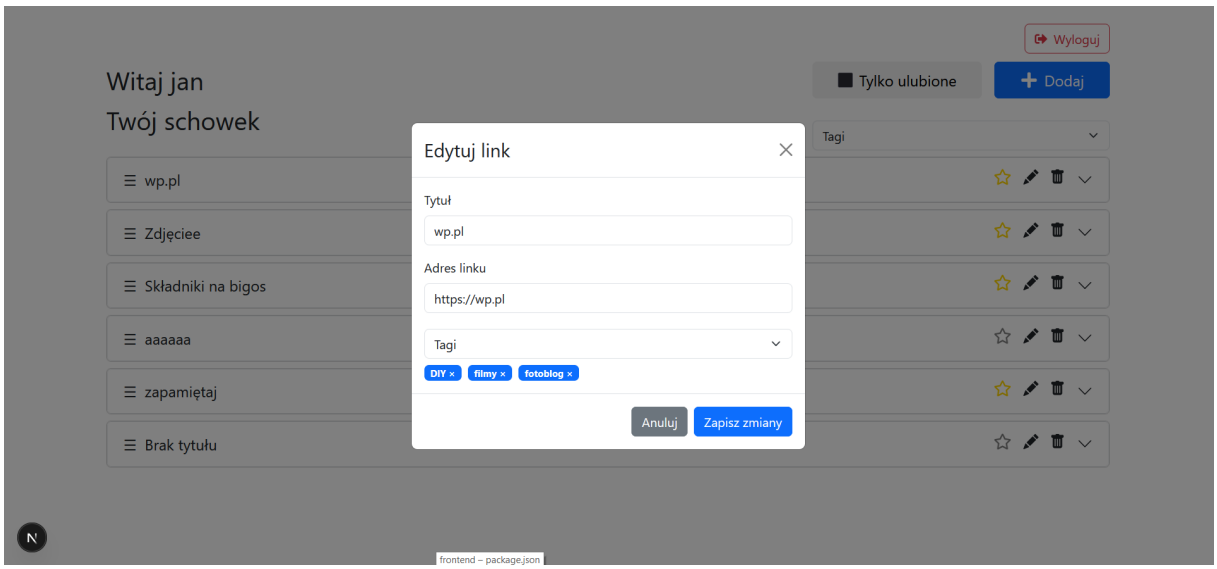
Rysunek 7: Dodawanie zasobu

6.4.3. Usuwanie zasobu



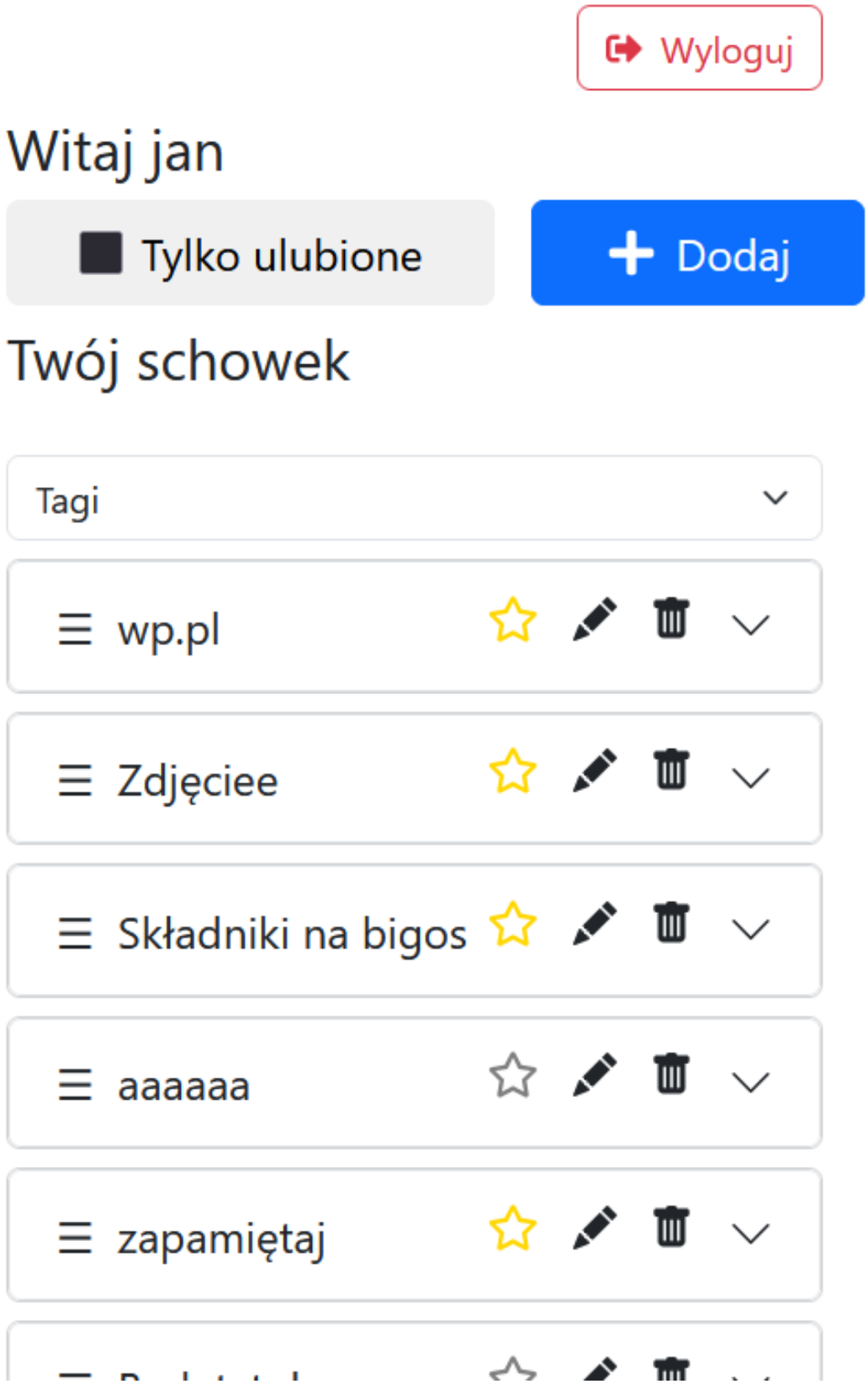
Rysunek 8: Usuwanie zasobu

6.4.4. Edycja zasobu



Rysunek 9: Edycja zasobu

6.5. Widok mobliny



Rysunek 10: Widok mobilny

7 Wnioski i dalszy rozwój

Projekt został zrealizowany w 100% i znacznie przekroczył początkowe założenia. Aplikacja jest w stanie produkcyjnym.

7.1. Podsumowanie

Projekt pokazał, że w ramach pracy studenckiej możliwe jest stworzenie nowoczesnej aplikacji webowej do przechowywania i organizowania obrazów, notatek oraz linków, z zastosowaniem aktualnych technologii i dobrych praktyk w inżynierii oprogramowania.

Kod źródłowy projektu jest dostępny w publicznym repozytorium na GitHub:
https://github.com/SulMorski/DataBox

Bibliografia

[1] Microsoft, ".net 8 documentation," https://learn.microsoft.com/pl-pl/dotnet/, 2025, dostęp: 15.10.2025.

[2] Vercel Inc., "Next.js documentation," https://nextjs.org, 2025, dostęp: 15.10.2025.

[3] Meta Platforms, "React documentation," https://react.dev, 2025, dostęp: 15.10.2025.

[4] Microsoft, "Typescript documentation," https://www.typescriptlang.org/docs/, 2025, dostęp: 18.11.2025.

[5] The PostgreSQL Global Development Group, "Postgresql 15 documentation," https://www.postgresql.org/docs/15/index.html, 2024.

[6] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston: Prentice Hall, 2017.

[7] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.