

# System Rekomendacji Książek

Dokumentacja Projektu

Przetwarzanie Danych w Chmurze Obliczeniowej

Julia Przeździk

Grudzień 2025

## Spis treści

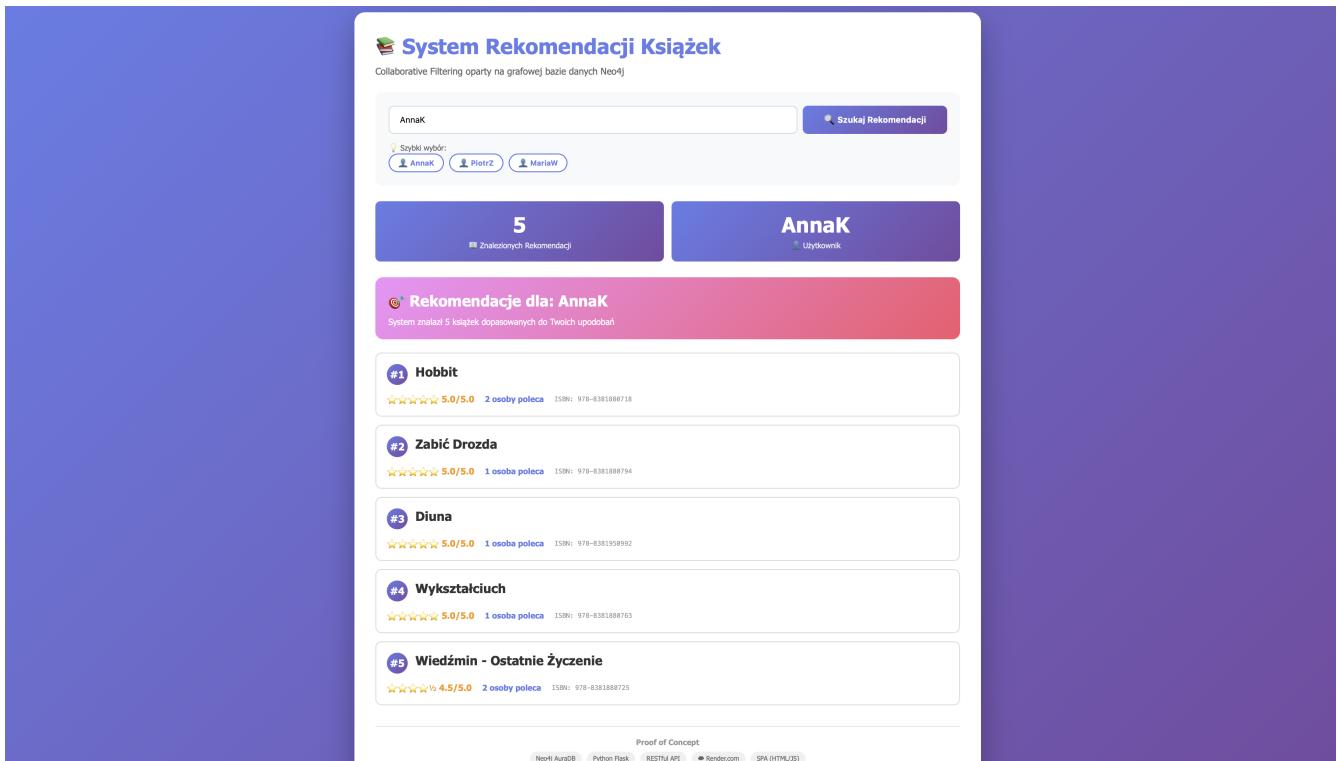
<b>1</b>	<b>Informacje Ogólne</b>	<b>3</b>
1.1	Cel Projektu . . . . .	3
<b>2</b>	<b>Prezentacja działania projektu</b>	<b>3</b>
2.1	Technologie . . . . .	5
2.2	Linki . . . . .	5
<b>3</b>	<b>Architektura Systemu</b>	<b>5</b>
3.1	Diagram Architektury . . . . .	5
3.2	Model Danych . . . . .	5
3.3	Struktura grafu w Neo4j . . . . .	6
3.4	Inicjalizacja modelu danych . . . . .	6
3.5	Wynik zapytania w Neo4j . . . . .	7
3.6	Algorytm rekomendacji . . . . .	7
<b>4</b>	<b>Implementacja</b>	<b>8</b>
4.1	Backend - Flask API . . . . .	8
4.2	Algorytm Collaborative Filtering . . . . .	8
4.3	Frontend - SPA . . . . .	8
<b>5</b>	<b>Testy i Walidacja</b>	<b>8</b>
5.1	Graph Database vs SQL . . . . .	8
<b>6</b>	<b>Instrukcja Uruchomienia</b>	<b>9</b>

# 1 Informacje Ogólne

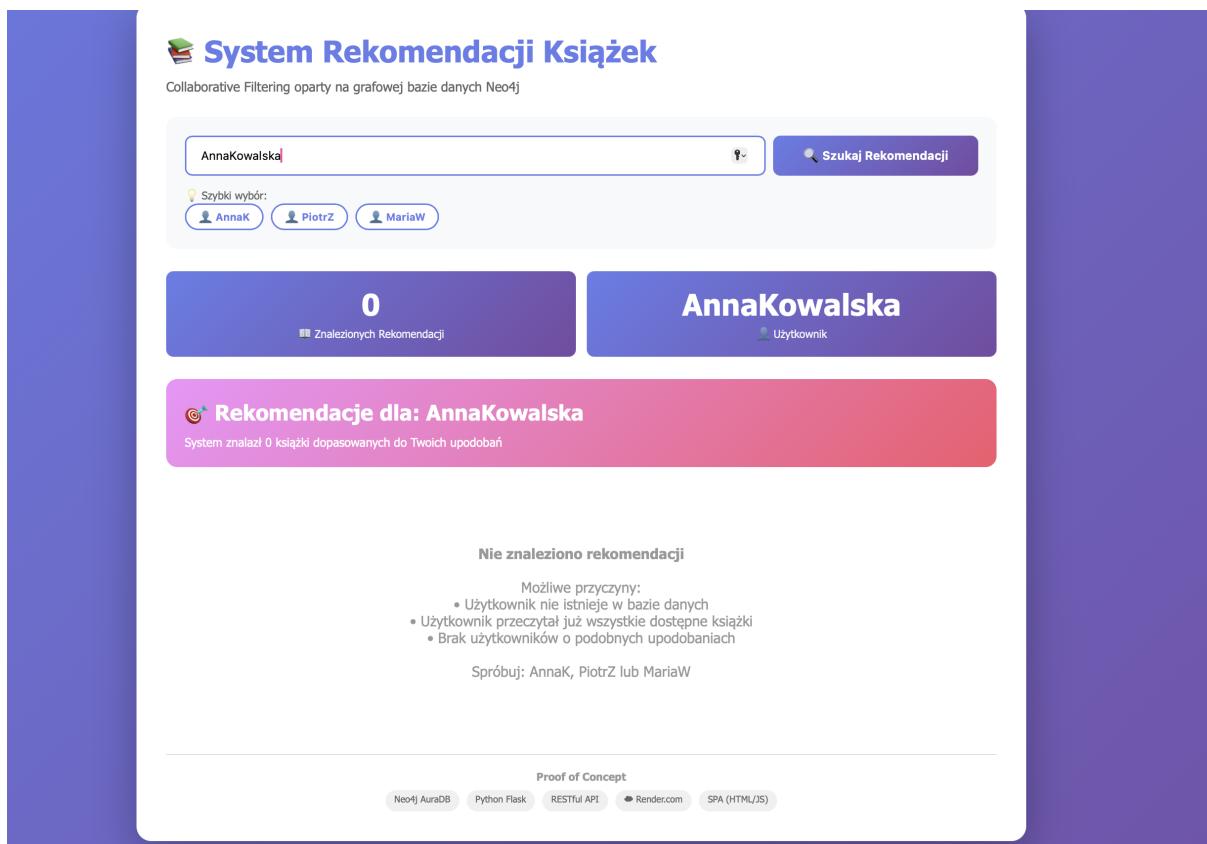
## 1.1 Cel Projektu

Celem projektu jest stworzenie systemu rekomendacji książek opartego o algorytm **Collaborative Filtering** wykorzystującego grafową bazę danych Neo4j. System analizuje preferencje użytkowników i na podstawie podobieństw proponuje książki.

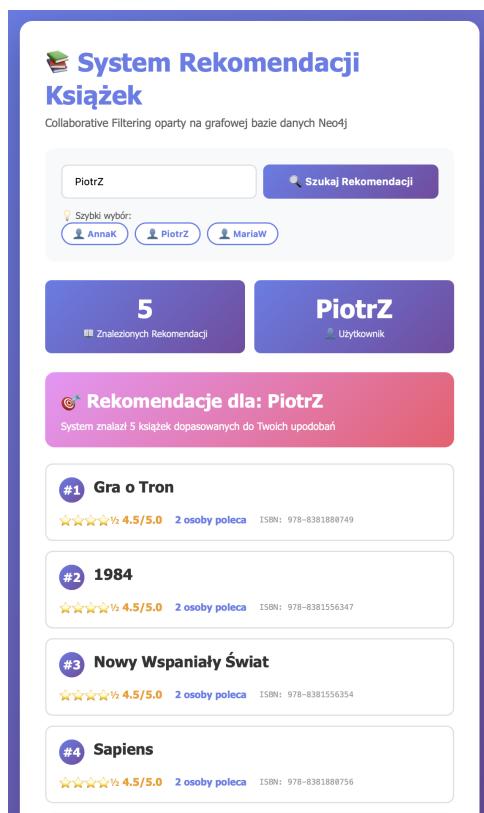
# 2 Prezentacja działania projektu



Rysunek 1: Strona podczas wyszukiwania osoby, której chcemy zarekomendować książkę



Rysunek 2: Wyniki podczas próby wyszukania osoby nie znajdującej się w bazie



Rysunek 3: Wersja responsywna

## 2.1 Technologie

### Stack Technologiczny

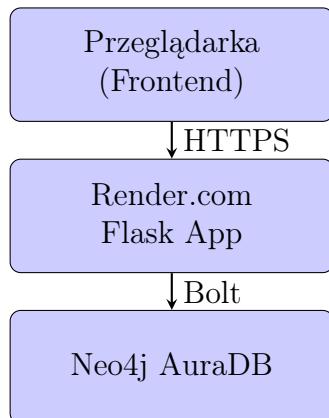
- **Backend:** Python 3.9 + Flask (RESTful API)
- **Baza danych:** Neo4j AuraDB Professional (DBaaS)
- **Frontend:** SPA - HTML5, CSS3, JavaScript ES6
- **Deployment:** Render.com (Cloud PaaS)

## 2.2 Linki

- **Demo:** <https://pdwcho-project.onrender.com>
- **GitHub:** [https://github.com/juliaprzezdzik/pdwcho\\_project.git](https://github.com/juliaprzezdzik/pdwcho_project.git)

## 3 Architektura Systemu

### 3.1 Diagram Architektury



Rysunek 4: Architektura systemu

### 3.2 Model Danych

#### Schemat Grafowy

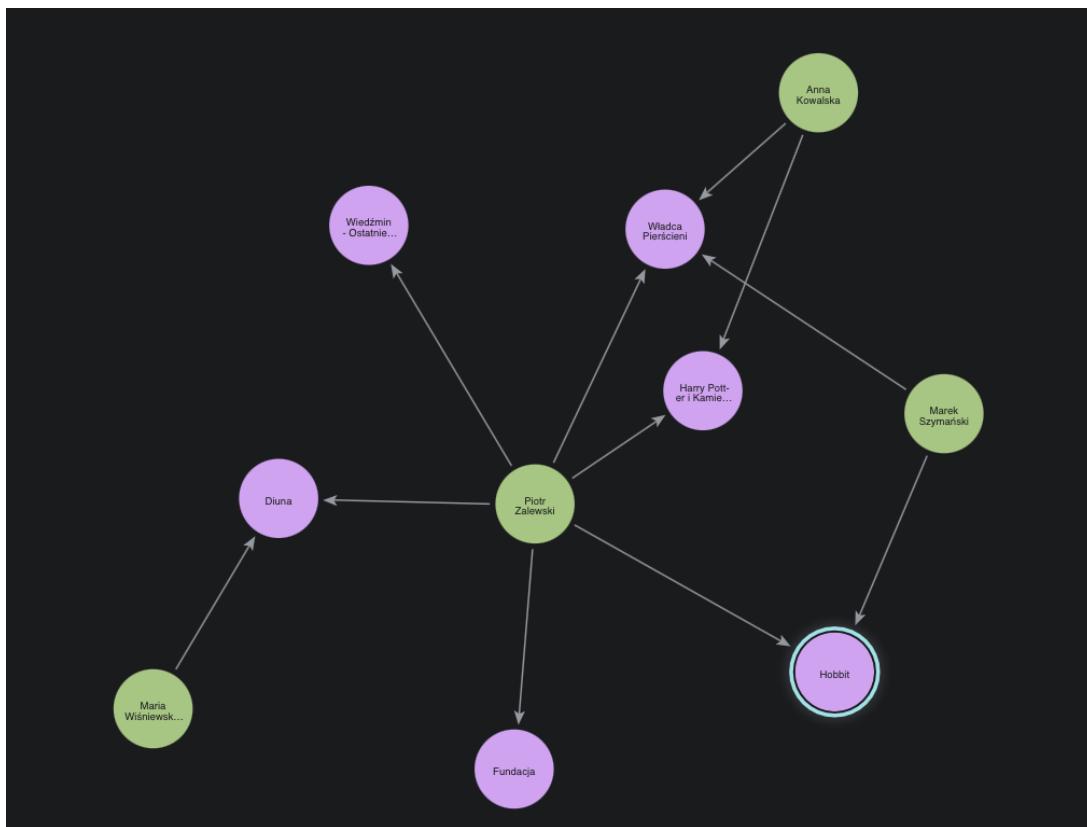
##### Węzły:

- **User:** username, name
- **Book:** isbn, title, author

##### Relacje:

- [:READS {rating: Float}]

### 3.3 Struktura grafu w Neo4j



### 3.4 Inicjalizacja modelu danych

```

1 MATCH (n) DETACH DELETE n;
2
3 // Użytkownicy
4 CREATE (anna:User {username: 'AnnaK'});
5 CREATE (piotr:User {username: 'PiotrZ'});
6 CREATE (maria:User {username: 'MariaW'});
7
8 // Książki
9 CREATE (hp:Book {title: 'Harry Potter', isbn: '123'});
10 CREATE (lotr:Book {title: 'Władca Pierścieni', isbn: '456'});
11 CREATE (hobbit:Book {title: 'Hobbit', isbn: '789'});
12 CREATE (orwell:Book {title: '1984', isbn: '101'});
13 CREATE (dune:Book {title: 'Diuna', isbn: '202'});
14
15 // Relacje - Anna
16 MATCH (anna:User {username: 'AnnaK'}), (hp:Book {title: 'Harry Potter'})
17 CREATE (anna)-[:READS {rating: 5}]->(hp);
18
19 MATCH (anna:User {username: 'AnnaK'}), (lotr:Book {title: 'Władca
    Pierścieni'})
20 CREATE (anna)-[:READS {rating: 4}]->(lotr);
21
22 // Relacje - Piotr
23 MATCH (piotr:User {username: 'PiotrZ'}), (hp:Book {title: 'Harry Potter
    '})
24 CREATE (piotr)-[:READS {rating: 5}]->(hp);
25
26 MATCH (piotr:User {username: 'PiotrZ'}), (lotr:Book {title: 'Władca
    Pierścieni'})
```

```

27 CREATE (piotr)-[:READS {rating: 4}]->(lotr);
28
29 MATCH (piotr:User {username: 'PiotrZ'}), (hobbit:Book {title: 'Hobbit'})
30 CREATE (piotr)-[:READS {rating: 5}]->(hobbit);
31
32 MATCH (piotr:User {username: 'PiotrZ'}), (dune:Book {title: 'Diuna'})
33 CREATE (piotr)-[:READS {rating: 4}]->(dune);
34
35 // Relacje - Maria
36 MATCH (maria:User {username: 'MariaW'}), (orwell:Book {title: '1984'})
37 CREATE (maria)-[:READS {rating: 5}]->(orwell);
38
39 MATCH (maria:User {username: 'MariaW'}), (dune:Book {title: 'Diuna'})
40 CREATE (maria)-[:READS {rating: 3}]->(dune);

```

### 3.5 Wynik zapytania w Neo4j

The screenshot shows the Neo4j browser interface with a query results table. The query is:

```

1 MATCH (u:User)-[r:READS]->(b:Book)
2 RETURN u.username, b.title, r.rating

```

The table has three columns: u.username, b.title, and r.rating. The data is as follows:

u.username	b.title	r.rating
<sup>1</sup> "AnnaK"	"Harry Potter"	5
<sup>2</sup> "PiotrZ"	"Harry Potter"	5
<sup>3</sup> "AnnaK"	"Władca Piersicieni"	4
<sup>4</sup> "PiotrZ"	"Władca Piersicieni"	4
<sup>5</sup> "PiotrZ"	"Hobbit"	5
<sup>6</sup> "MariaW"	"1984"	5
<sup>7</sup> "PiotrZ"	"Diuna"	4
<sup>8</sup> "MariaW"	"Diuna"	3

### 3.6 Algorytm rekomendacji

```

1 MATCH (me:User {username: $username}) -[:READS]->(b:Book) <-[:READS]-(other:User)
2 WHERE other <> me
3 WITH other, COUNT(DISTINCT b) AS commonBooks
4 ORDER BY commonBooks DESC
5 LIMIT 5
6
7 MATCH (other)-[r:READS]->(recBook:Book)
8 WHERE NOT EXISTS {
9     MATCH (me:User {username: $username}) -[:READS]->(recBook)

```

```

10 }
11
12 RETURN recBook.title AS title,
13     recBook.isbn AS isbn,
14     AVG(r.rating) AS avgRating,
15     COUNT(other) AS recommenders
16 ORDER BY avgRating DESC, recommenders DESC
17 LIMIT 5

```

## 4 Implementacja

### 4.1 Backend - Flask API

Metoda	Endpoint	Opis
GET	/	Strona główna
GET	/hello	Healthcheck
GET	/recommendations/<user>	Rekomendacje

Tabela 1: Endpointy API

### 4.2 Algorytm Collaborative Filtering

1. Znajdź podobnych użytkowników - według wspólnych książek
2. Oblicz similarity score - liczba wspólnych książek
3. Generuj rekomendacje - książki podobnych użytkowników
4. Ranking - sortuj według oceny i popularności

### 4.3 Frontend - SPA

Funkcjonalności:

Responsywny interfejs

Animacje CSS3

Error handling

## 5 Testy i Walidacja

### 5.1 Graph Database vs SQL

SQL (JOIN-heavy):

```

1 SELECT b.title FROM users u1
2 JOIN reads r1 ON u1.id = r1.user_id
3 JOIN reads r2 ON r1.book_id = r2.book_id
4 JOIN users u2 ON r2.user_id = u2.id
5 WHERE u1.username = 'AnnaK'

```

Test Case		Input	Output	Status
Rekomendacje kownika	użytkownika	AnnaK	5 książek	
Nieistniejący kownik	użytkownik	XYZ	Pusta lista	
Healthcheck		/hello	Status 200	
Responsiveness		375px	OK layout	

Tabela 2: Testy funkcjonalne

### Neo4j Cypher:

```

1 MATCH (me:User {username: 'AnnaK'}) -[:READS] ->()
2   <-[:READS]-(other)
3 MATCH (other)-[:READS]->(recBook)
4 RETURN recBook.title

```

## 6 Instrukcja Uruchomienia

```

1 # Clone repository
2 git clone https://github.com/username/PDCHO_PROJEKT.git
3 cd PDCHO_PROJEKT
4
5 # Virtual environment
6 python3 -m venv venv
7 source venv/bin/activate
8
9 # Instalacja
10 pip install -r requirements.txt
11
12 # Zmienne środowiskowe
13 export NEO4J_URI="neo4j+s://... "
14 export NEO4J_USER="neo4j"
15 export NEO4J_PASSWORD="... "
16
17 # Uruchomienie
18 python app.py

```

## Podsumowanie

Projekt prezentuje system rekomendacji wykorzystujący Neo4j i nowoczesne technologie chmurowe.