

# **MOVIE SEARCHER**

## **SYSTEM DO WYSZUKIWANIA FILMÓW**

Gabriela Rostek 57077

Natalia Olejnik 55244

Jakub Kiełb 51437

Sebastian Kowalski 52664

Jakub Kulik 56201

## Spis treści

1.	Ogólne założenia systemu .....	3
2.	Diagramy .....	4
2.1	Diagram aktywności .....	4
2.2	Diagram sekwencji .....	5
2.3	Diagram Use Case .....	6
2.3.1	Opis przypadku użycia – wyszukaj film .....	6
2.4	Diagram klas .....	7
3.	Wymagania systemu .....	7
3.1	Wymagania funkcjonalne .....	7
4.	Testy .....	9
5.	Projekt interfejsu użytkownika .....	12
5.1	Panel rejestracji nowego użytkownika .....	12
5.2	Panel logowania użytkownika .....	12
5.3	Panel zmiany hasła .....	13
5.4	Panel wyszukiwania filmu .....	13
6.	Przyszła ewolucja systemu .....	14
7.	Historia zmian projektu .....	15

# 1. Ogólne założenia systemu

Celem systemu jest umożliwienie użytkownikowi sprawne wyszukanie danego filmu na podstawie jego nazwy. Po wpisaniu tytułu lub części frazy oraz zaakceptowania wyszukiwania, system ma za zadanie wyszukać pożądany film i wyświetlić użytkownikowi jego szczegóły: tytuł filmu, plakat filmu, data produkcji filmu oraz gatunek filmowy z którego się wywodzi.

Backend systemu jest napisany w języku Python ze względu na wysokie możliwości tego języka w dzisiejszych czasach, w przyszłości z łatwością będzie możliwość rozbudowania systemu o nowe funkcje.

Baza danych stworzona została korzystając z podejścia database first, ze względu na małą złożoność relacyjnej bazy danych, która nie wymaga żmudnego tworzenia dużej ilości tabel oraz relacji. Baza danych została napisana za pomocą MySQL.

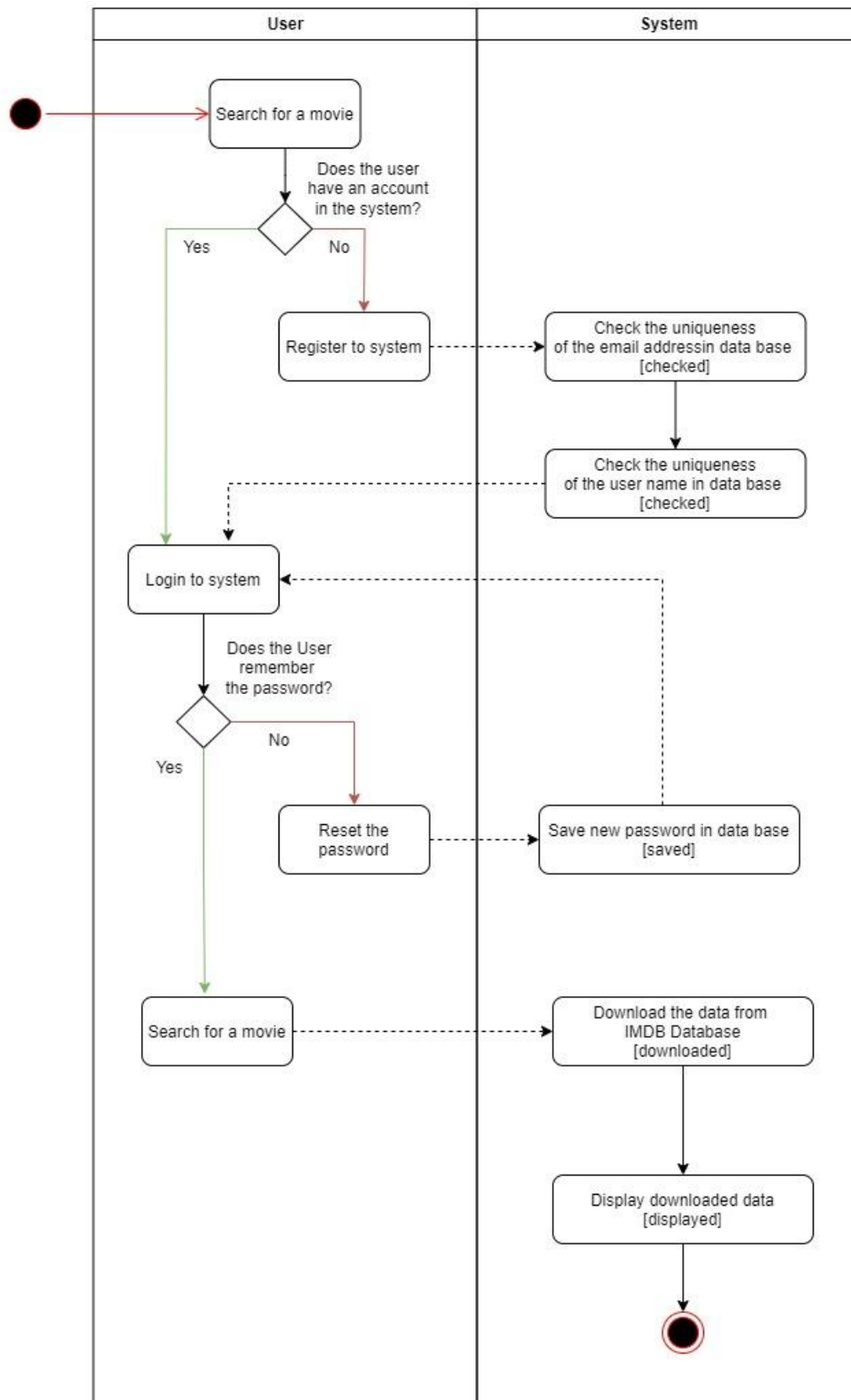
Frontend aplikacji webowej napisany został w JavaScript, CSS oraz HTML, połączenie tych języków daje ogromne możliwości, w tym segmencie swoje miejsce znalazło również API, które łączy się i pobiera dane z bazy danych IMDB.

Prototyp aplikacji webowej, który określił wstępny wygląd oraz podstawowe funkcjonalności wykonano za pomocą narzędzia Figma.

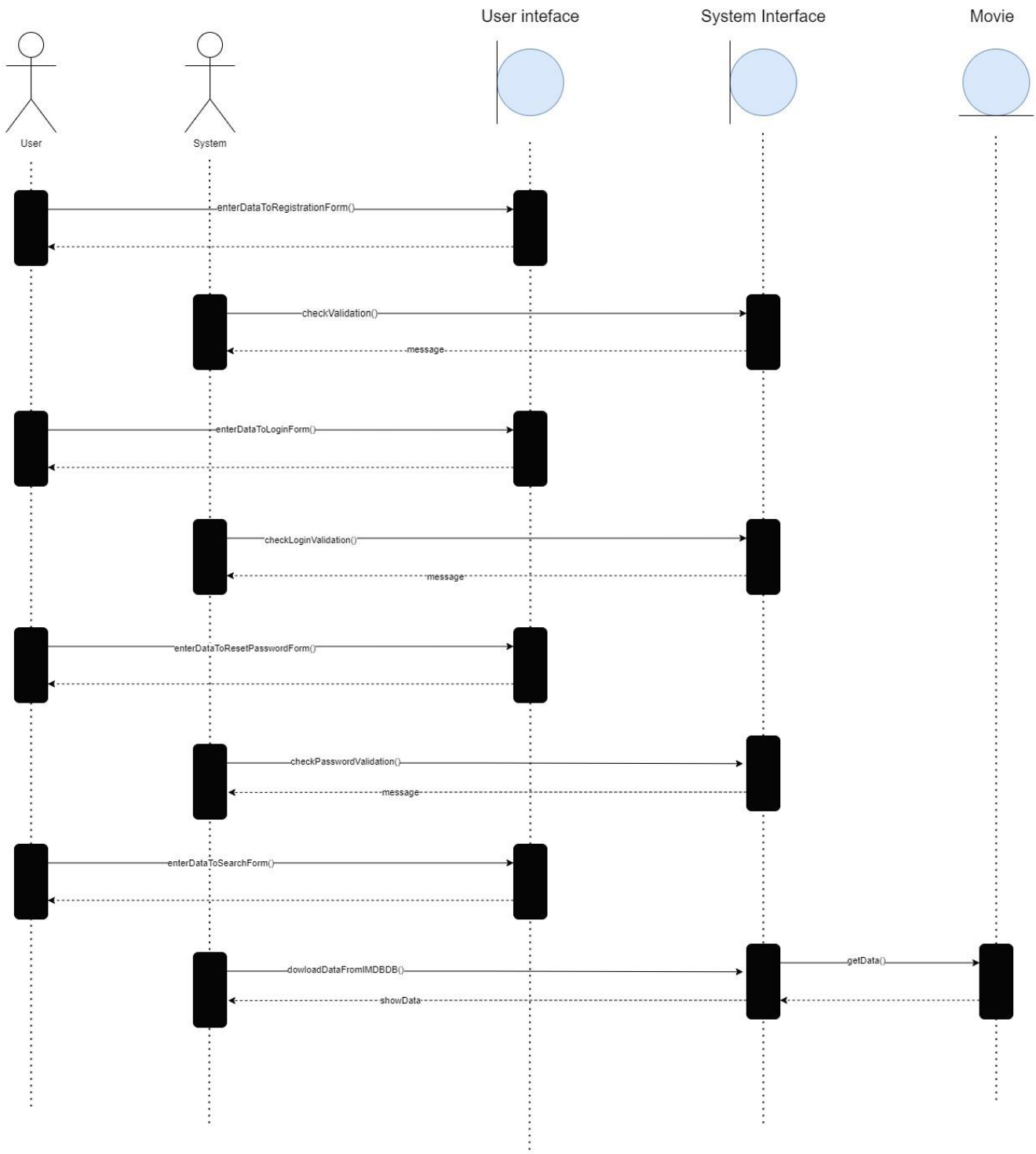
Współpraca zespołu podczas tworzenia systemu była oparta na umieszczaniu swojej części pracy na współdzielonym repozytorium Github.

## 2. Diagramy

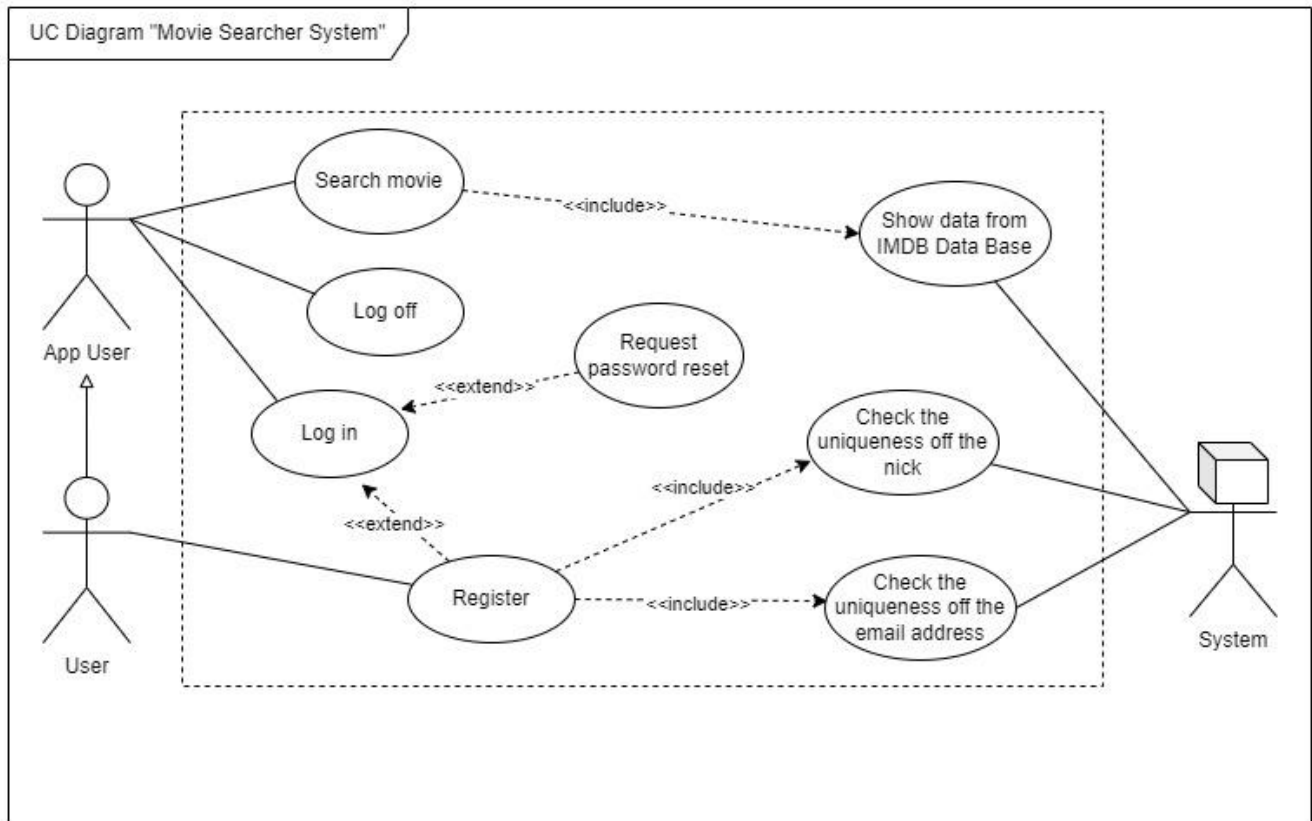
### 2.1 Diagram aktywności



## 2.2 Diagram sekwencji



## 2.3 Diagram Use Case



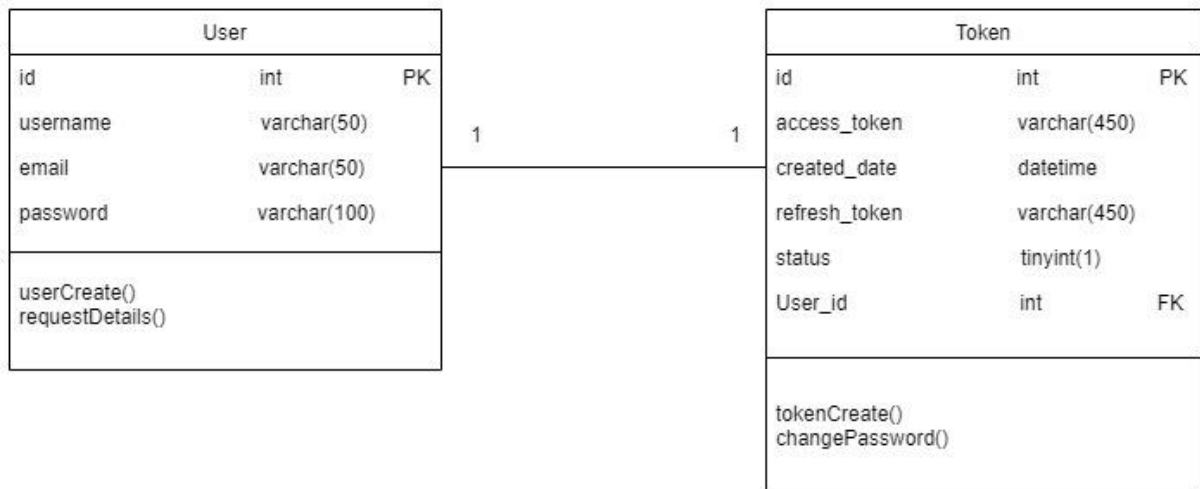
### 2.3.1 Opis przypadku użycia – wyszukaj film

Użytkownik tworzy konto w systemie, a po rejestracji przechodzi do logowania. Po pomyślnym zalogowaniu do systemu, użytkownikowi ujawnią się nowe opcje takie jak wyszukaj film oraz wyloguj.

Użytkownik systemu wpisuje tytuł lub jego część i zatwierdza wyszukiwanie. Po zatwierdzeniu system ma za zadanie wyszukać filmy, które spełniają kryteria wyszukiwania w części lub w całości, następnie pobrać dane o nich z zewnętrznej bazy danych i wyświetlić do wglądu użytkownika.

Przypadek użycia kończy się.

## 2.4 Diagram klas



## 3. Wymagania systemu

### 3.1 Wymagania funkcjonalne

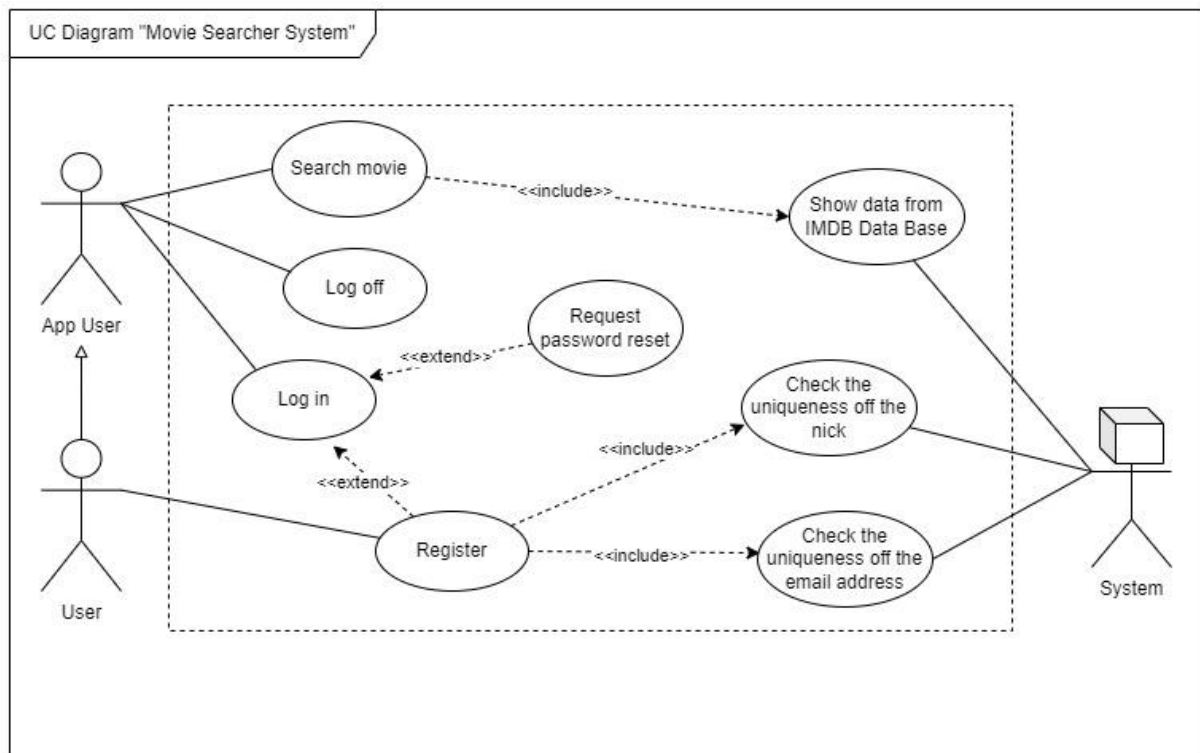
Użytkownik powinien mieć możliwość:

- założenia konta za pomocą własnych danych (nazwa użytkownika, email i hasło),
- zalogowania się do systemu po wcześniejszej rejestracji za pomocą nazwy użytkownika i hasła,
- resetowania hasła,
- wyszukania filmu,
- wylogowania się z systemu.

System powinien:

- walidować prowadzone przez użytkownika dane (unikalność nazwy użytkownika oraz email w bazie danych systemu),
- pobierać i wyświetlać dane dotyczące wyszukiwanych filmów z bazy danych IMDB,
- aktualizować dane użytkownika w bazie danych.

Wymagania funkcjonalne najlepiej obrazuje diagram Use Case:



### 3.2 Wymaganie niefunkcjonalne

System do wyszukiwania filmów „Movie Search” powinien działać przy następujących ograniczeniach:

- Dane znajdujące się w bazie danych powinny być dostępne 24h na dobę 7 dni w tygodniu,
- Przerwy techniczne w celu kontroli systemu lub wprowadzenia zmian powinny odbywać się między godziną 1:00 a 6:00,
- Przeglądarka internetowa Google Chrome, Microsoft Edge, Firefox,
- Minimalna ilość pamięci ram to 4GB,
- System powinien być zabezpieczony przed nieautoryzowanym dostępem,
- Hasła użytkowników powinny być szyfrowane,
- System powinien być zgodny ze wszystkimi aktualnie obowiązującymi przepisami i prawami,
- Interfejs graficzny systemu powinien być intuicyjny,
- Backend aplikacji – Python,
- Baza Danych – MySQL,
- Frontend aplikacji – JavaScript.



## 4. Testy

Napisane testy jednostkowe i integracyjne obejmują testowanie rejestracji i logowania użytkownika, zmianę hasła oraz samo wylogowanie z systemu. Testy polegają na tworzeniu dwóch użytkowników i przetestowanie na nich funkcjonalności systemu, jeżeli rezultat testów będzie pozytywny skrypt uruchamia frontend, w przeciwnym przypadku cały system wyłącza się z odpowiednim kodem błędu.

Definiowanie:

```
def generate_random_username():
    return "test_user_" + str(uuid.uuid4()).replace("-", "")[:8]

def generate_random_email():
    return "test_" + str(uuid.uuid4()).replace("-", "")[:8] + "@example.com"

def generate_random_password():
    return "password123"

@pytest.fixture(scope="module")
def user_data_1():
    username = generate_random_username()
    email = generate_random_email()
    password = generate_random_password()
    return {"username": username, "email": email, "password": password}

@pytest.fixture(scope="module")
def new_user_1(user_data_1):
    response = requests.post(f"{BASE_URL}/register", json=user_data_1)
    assert response.status_code == 200
    return user_data_1

@pytest.fixture(scope="module")
def logged_in_user_1(new_user_1):
    username = new_user_1["username"]
    password = new_user_1["password"]
    login_data = {"username": username, "password": password}
    response = requests.post(f"{BASE_URL}/login", json=login_data)
    assert response.status_code == 200
    return response.json()["access_token"]

@pytest.fixture(scope="module")
def user_data_2():
    username = generate_random_username()
    email = generate_random_email()
    password = generate_random_password()
    return {"username": username, "email": email, "password": password}
```

```

@pytest.fixture(scope="module")
def new_user_2(user_data_2):
    response = requests.post(f"{BASE_URL}/register", json=user_data_2)
    assert response.status_code == 200
    return user_data_2

@pytest.fixture(scope="module")
def logged_in_user_2(new_user_2):
    username = new_user_2["username"]
    password = new_user_2["password"]
    login_data = {"username": username, "password": password}
    response = requests.post(f"{BASE_URL}/login", json=login_data)
    assert response.status_code == 200
    return response.json()["access_token"]

```

Test resetowania hasła przez użytkownika:

```

def test_forgot_password_1(user_data_1):
    username = user_data_1["username"]
    new_password = generate_random_password()
    password_data = {"username": username, "new_password": new_password,
"repeat_new_password": new_password}
    response = requests.put(f"{BASE_URL}/forgot-password", json=password_data)
    assert response.status_code == 200
    assert response.json()["message"] == "Password changed successfully"

def test_forgot_password_2(user_data_2):
    username = user_data_2["username"]
    new_password = generate_random_password()
    password_data = {"username": username, "new_password": new_password,
"repeat_new_password": new_password}
    response = requests.put(f"{BASE_URL}/forgot-password", json=password_data)
    assert response.status_code == 200
    assert response.json()["message"] == "Password changed successfully"

```

Test wylogowania użytkownika z systemu:

```
def test_logout_1(logged_in_user_1):
    headers = {"Authorization": f"Bearer {logged_in_user_1}"}
    response = requests.post(f"{BASE_URL}/logout", headers=headers)
    assert response.status_code == 200
    assert response.json()["message"] == "Logout Successfully"

def test_logout_2(logged_in_user_2):
    headers = {"Authorization": f"Bearer {logged_in_user_2}"}
    response = requests.post(f"{BASE_URL}/logout", headers=headers)
    assert response.status_code == 200
    assert response.json()["message"] == "Logout Successfully"
```

Test rejestracji użytkownika w systemie:

```
def test_register_1(new_user_1):
    assert new_user_1 is not None

def test_register_2(new_user_2):
    assert new_user_2 is not None
```

Test logowania użytkownika do systemu:

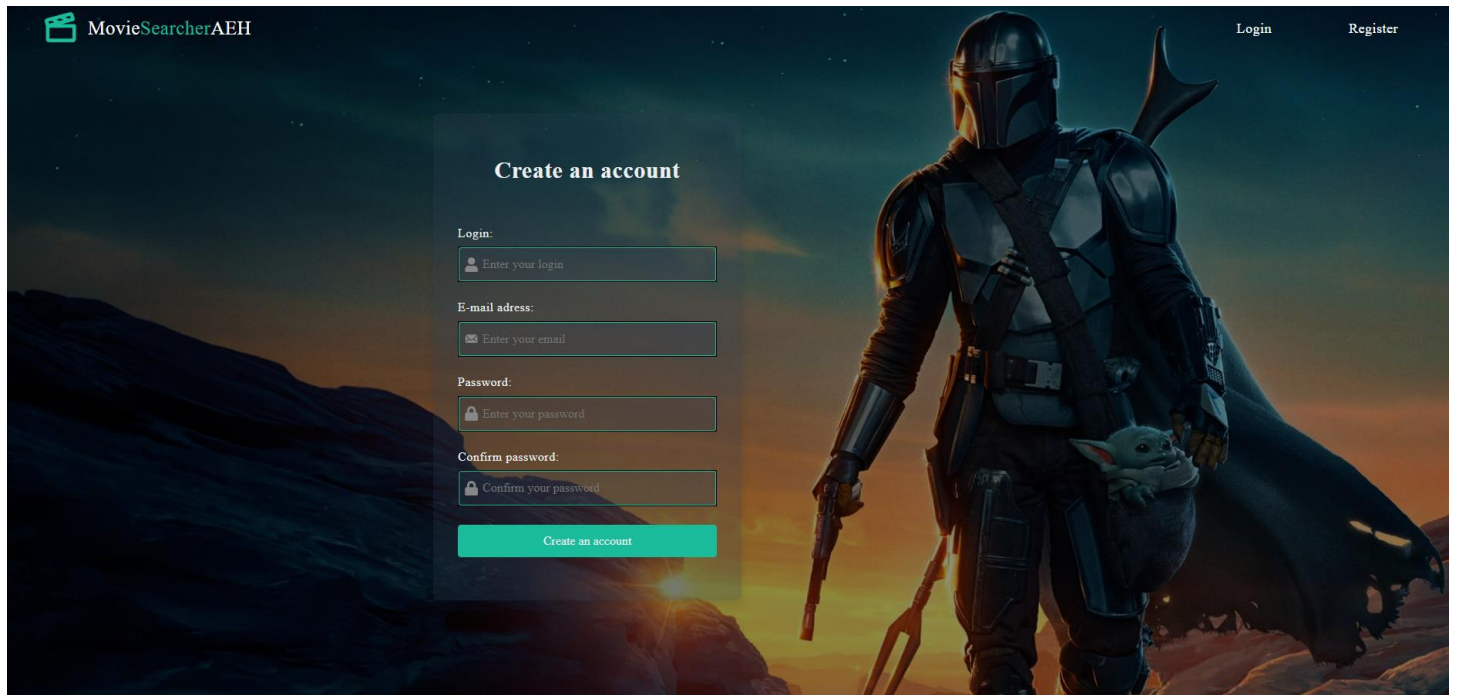
```
def test_login_1(logged_in_user_1):
    assert logged_in_user_1 is not None

def test_login_2(logged_in_user_2):
    assert logged_in_user_2 is not None
```

Przed każdorazowym uruchomieniu systemu skrypt uruchamia testy weryfikujące poprawność w działaniu funkcjonalności systemu. W przypadku występowania jakiegokolwiek błędu w działaniu system zwraca odpowiedni błąd.

## 5. Projekt interfejsu użytkownika

### 5.1 Panel rejestracji nowego użytkownika



MovieSearcherAEH

Login Register

### Create an account

Login:

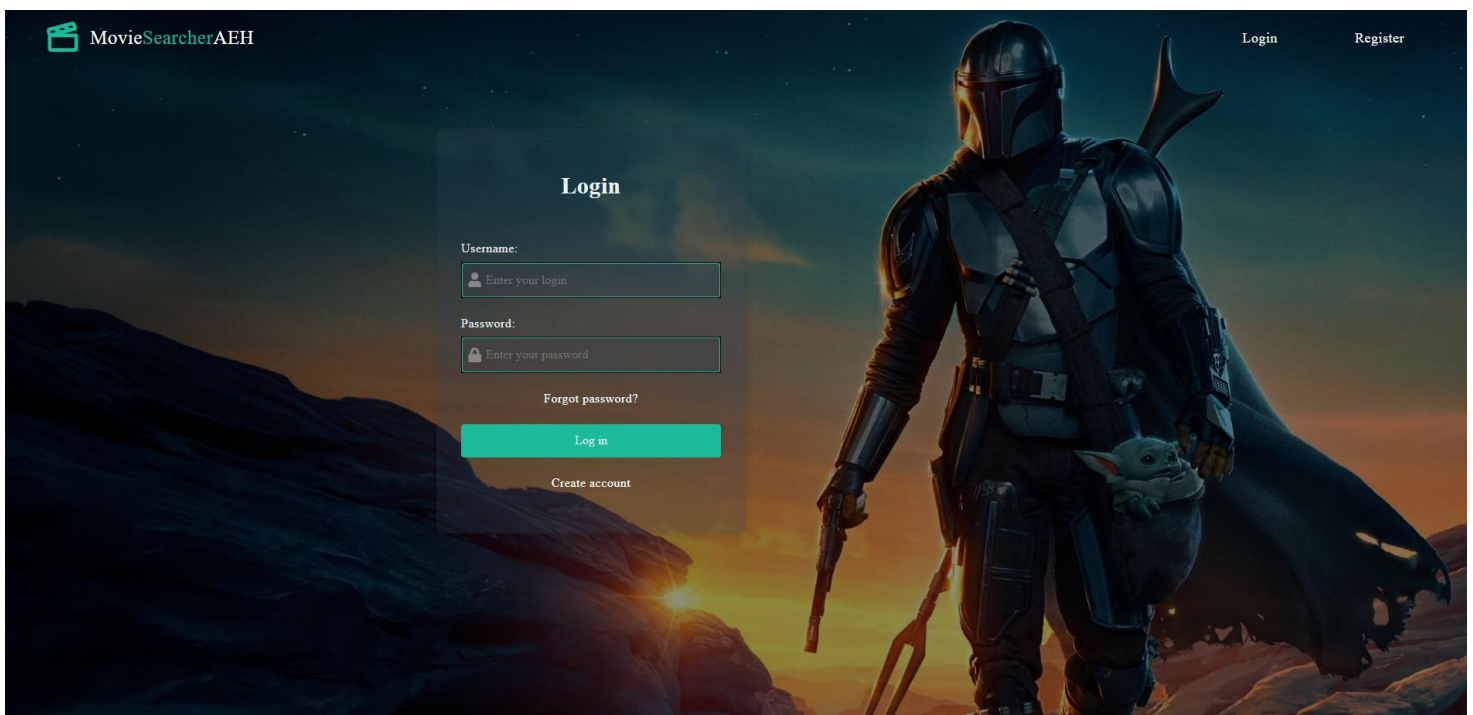
E-mail address:

Password:

Confirm password:

Create an account

### 5.2 Panel logowania użytkownika



MovieSearcherAEH

Login Register

### Login

Username:

Password:

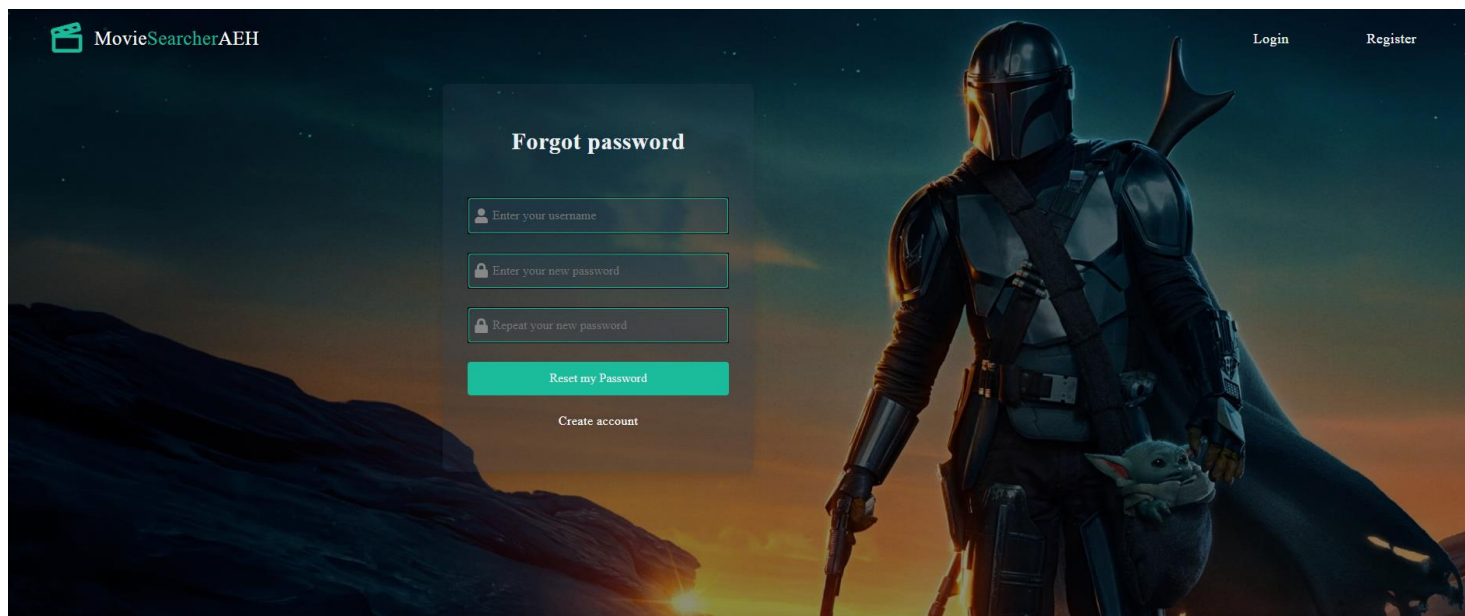
[Forgot password?](#)

Log in

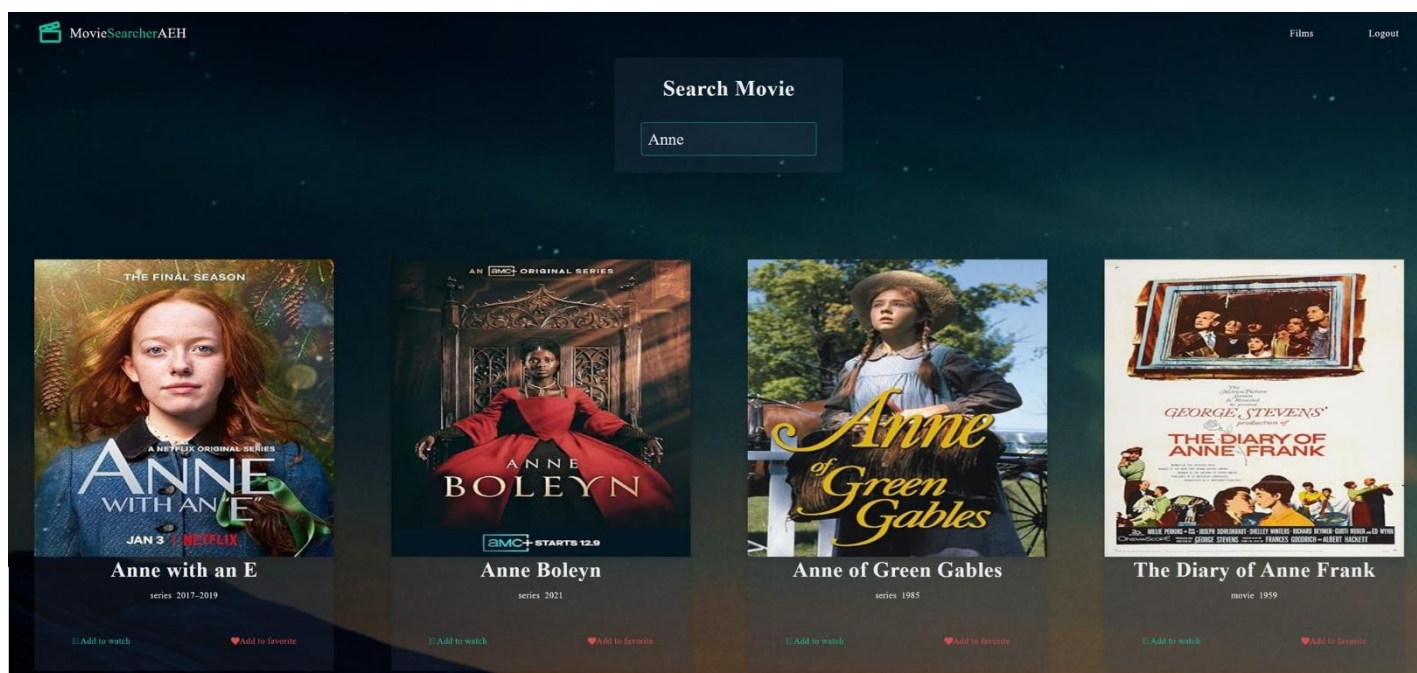
[Create account](#)



### 5.3 Panel zmiany hasła



### 5.4 Panel wyszukiwania filmu

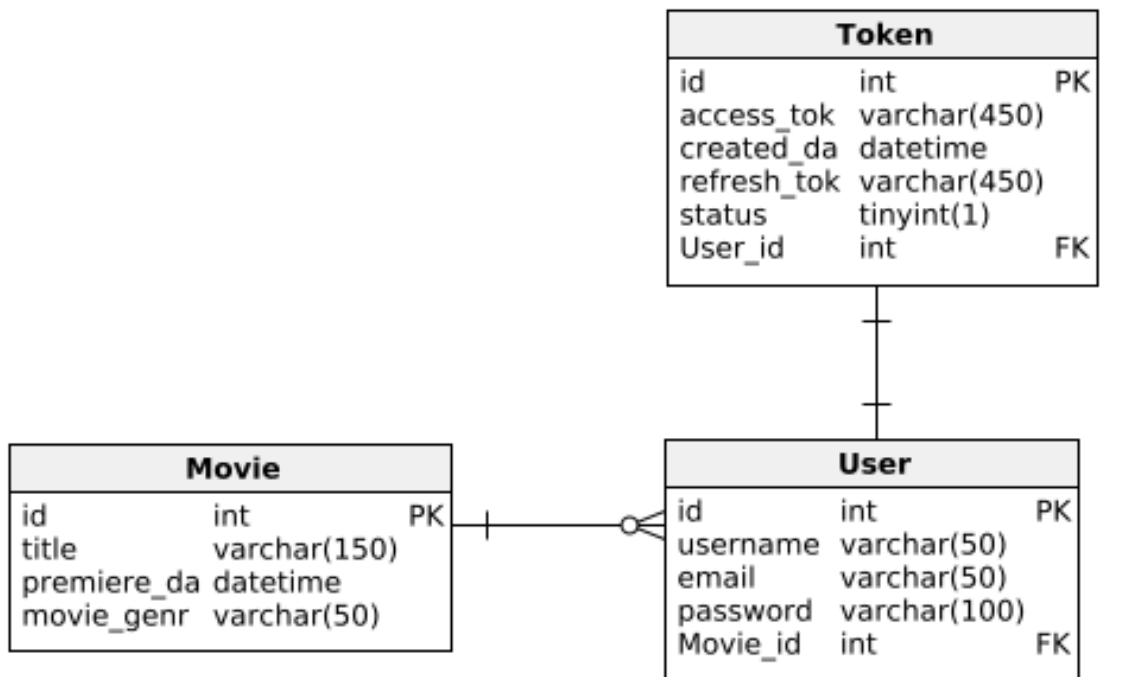


## 6. Przyszła ewolucja systemu

System mógłby być rozwinięty o możliwość tworzenia własnych bibliotek filmowych przez użytkowników. Użytkownik mógłby dodawać filmy do swojej kolekcji i odznaczyć w przypadku obejrzenia.

Kolejnym pomysłem na rozwinięcie systemu jest wprowadzenie podziału na Filmy oraz Seriale oraz wprowadzenie aplikacji na urządzenia mobilne, gdzie dodatkową funkcjonalnością byłoby wysyłanie powiadomienia użytkownikowi np. w przypadku pojawienia się nowego odcinka serialu lub pojawienia się danego filmu w telewizji, aby uniknąć opłat związanych z oglądaniem filmu na platformie streamingowej.

Przykładowy diagram klas dla rozwiniętego systemu:



## 7. Historia zmian projektu

Tabela historii zmian projektu:

Autor zmian	Data zmian	Opis zmian	Wersja
Gabriela Rostek, Natalia Olejnik, Jakub Kiełb, Sebastian Kowalski, Jakub Kulik	24.05.2024r	Stworzenie projektu i zaimplementowanie aplikacji webowej „Movie Searcher”	1.0