

EXPERT SYSTEM

1. Wprowadzenie

Opis i cel projektu:

Projekt to system ekspertowy, który wykorzystuje logiczny język programowania Prolog do definiowania warunków doradzania. Specyfika tego projektu polega na tym, że system jest stworzony jako ekspert doradzający w dziedzinie japońskich animacji.

Dzięki integracji z językiem Java i narzędziem Spring, stworzyliśmy interfejs użytkownika, który umożliwia interakcję z systemem. Użytkownik może zadawać pytania dotyczące różnych aspektów anime, a system na podstawie zgromadzonej wiedzy w bazie Prologa udziela odpowiedzi i doradza.

Centralnym elementem projektu związanym z językiem Prolog jest plik zawierający bazę wiedzy i zdefiniowane zapytania dotyczące anime. Poprzez korzystanie z tego pliku, system jest w stanie generować odpowiednie wyniki, dostarczając spersonalizowane rekomendacje i porady w tej dziedzinie.

Dzięki integracji obu języków przy użyciu Springa, projekt dostarcza wiele zalet. Możliwe jest łatwe zarządzanie zależnościami, obsługa żądań HTTP, testowanie oraz integracja z innymi modułami Springa, które mogą być wykorzystane w projekcie.

Wyniki, które system zwraca, oparte są na logice i regułach zdefiniowanych w bazie wiedzy opartej na Prologu. Dzięki temu użytkownik otrzymuje rzetelne i spersonalizowane porady dotyczące anime.

Technologie używane w projekcie:

- Prolog: Logiczny język programowania używany do definiowania warunków doradzania w systemie. Pozwala na zapisanie reguł, które system wykorzystuje do udzielania odpowiedzi na pytania dotyczące anime.
- Java: Język programowania, który integruje język Prolog z interfejsem użytkownika. Java zapewnia programowanie obiektowe i umożliwia zarządzanie logiką systemu oraz interakcję z użytkownikiem.
- Spring: Narzędzie i framework do aplikacji Java, używane do tworzenia interfejsu użytkownika. Spring oferuje funkcje takie jak zarządzanie zależnościami, obsługa żądań HTTP, testowanie i integracja z innymi modułami Springa. Użytkownik może łatwo komunikować się z systemem i zadawać pytania dotyczące anime dzięki Springowi.
- JPL (Java Prolog Interface): Biblioteka umożliwiająca komunikację między językiem Java a językiem Prolog. JPL została wykorzystana w projekcie do bezproblemowej integracji Prologa z aplikacją napisaną w Javie. Umożliwia przesyłanie zapytań do bazy wiedzy Prologa i otrzymywanie wyników w aplikacji Java.
- Thymeleaf: Silnik szablonów służący do tworzenia interfejsu użytkownika w aplikacjach webowych. Thymeleaf został użyty w projekcie do generowania dynamicznych stron HTML opartych na danych przekazywanych przez system. Pozwala na łatwe wstawianie danych i logiki w szablony HTML, co umożliwia tworzenie interaktywnego interfejsu użytkownika.

Główne funkcjonalności systemu:

- Doradzanie w dziedzinie anime: System oparty na języku Prolog jest w stanie udzielać użytkownikowi rekomendacji i porad dotyczących różnych aspektów anime. Na podstawie zgromadzonej wiedzy w bazie Prologa, system analizuje pytania użytkownika i generuje odpowiedzi dostosowane do indywidualnych preferencji.
- Interakcja użytkownika z systemem: Dzięki interfejsowi użytkownika opartemu na Springu i Thymeleaf, użytkownik może łatwo zadawać pytania dotyczące anime i otrzymywać odpowiedzi od systemu. Interfejs jest intuicyjny i przyjazny, umożliwiając użytkownikowi wygodne korzystanie z systemu.
- Personalizowane rekomendacje: System wykorzystuje zgromadzoną bazę wiedzy, aby dostarczać spersonalizowane

2. Architektura systemu:

Architektura systemu opiera się na integracji języka Prolog z aplikacją Java Spring przy użyciu biblioteki JPL7 (Java Prolog Interface). JPL7 umożliwia komunikację między językiem Java a językiem Prolog, co pozwala na przekazywanie zapytań do bazy wiedzy Prologa i otrzymywanie wyników w aplikacji Java.

Warstwy integracji Prologu z aplikacją Java Spring:

- Warstwa interfejsu użytkownika: Odpowiada za interakcję z użytkownikiem, przyjmowanie pytań i wyświetlanie odpowiedzi. W tej warstwie wykorzystywane są Java Spring i framework Thymeleaf do generowania dynamicznych stron HTML.
- Warstwa logiki biznesowej: Zajmuje się przetwarzaniem pytań użytkownika i przekazywaniem ich do modułu Prologu. W tej warstwie znajduje się integracja JPL7, która umożliwia przesyłanie zapytań do bazy wiedzy Prologa i otrzymywanie wyników.
- Warstwa bazodanowa: Zawiera plik Prologu (baza.pl) z zdefiniowaną bazą wiedzy o anime. Ten plik Prologu jest wykorzystywany przez moduł Prologu do udzielania odpowiedzi na pytania.

Integracja Prolog z aplikacją Java Spring:

Integracja języka Prolog z aplikacją Java Spring odbywa się przy użyciu JPL7, który umożliwia bezproblemową komunikację między językiem Prolog a językiem Java. JPL7 działa jako mostek między dwoma językami, umożliwiając przekazywanie zapytań i wyników.

W aplikacji Java Spring, moduł Prologu jest inicjalizowany i konfigurowany przy użyciu JPL7. Tworzony jest obiekt z interfejsem Prologu, który umożliwia przekazywanie zapytań do bazy wiedzy Prologa i otrzymywanie wyników.

Pytania od użytkownika przekazywane są do modułu Prologu poprzez wywołanie odpowiednich metod interfejsu Prologu. Moduł Prologu przetwarza zapytania, wykonuje wnioskowanie na podstawie bazy wiedzy Prologa i zwraca wyniki do aplikacji Java Spring.

Aplikacja Java Spring może następnie przetworzyć otrzymane wyniki i wyświetlić odpowiedzi użytkownikowi na interfejsie użytkownika.

Integracja języka Prolog z aplikacją Java Spring za pośrednictwem JPL7 umożliwia wykorzystanie potencjału obu języków w tworzeniu systemu ekspertowego doradającego w dziedzinie anime.

3. Opis pliku wykorzystującego prolog

Czym jest język prolog i dlaczego się go używa w systemach ekspertowych

Język Prolog jest wykorzystywany w systemach ekspertowych ze względu na swoją zdolność do wyrażania wiedzy w sposób logiczny. Dzięki programowaniu przez logikę i mechanizmowi unifikacji, Prolog umożliwia automatyczne wnioskowanie na podstawie faktów i reguł. Jest efektywny w przetwarzaniu zapytań i umożliwia precyzyjne modelowanie wiedzy ekspertów.

Baza wiedzy

To zbiór faktów które wspólnie tworzą ową bazę są one zapewnianiem informacji na bazie których program w zapytaniach podejmuje decyzje. (fragmenty z 3 faktami bazy poniżej)

```
anime(32281, 'Kimi no Na wa.', ['Drama', 'Romance', 'School', 'Supernatural'], 'Movie', 1, 9.37, 200630).
anime(5114, 'Fullmetal Alchemist: Brotherhood', ['Action', 'Adventure', 'Drama', 'Fantasy', 'Magic', 'Military', 'Shounen'], 'TV', 64, 9.26, 793665).
anime(28977, 'Gintama', ['Action', 'Comedy', 'Historical', 'Parody', 'Samurai', 'Sci-Fi', 'Shounen'], 'TV', 51, 9.25, 114262).
```

Różnica pomiędzy bazą wiedzy i bazą danych

Baza wiedzy jest reprezentacją logiczną informacji i reguł, umożliwiającą wnioskowanie i przetwarzanie wiedzy. Baza danych natomiast jest strukturą przechowującą dane w formie tabel, umożliwiającą efektywne ich przechowywanie, zapytywanie i manipulację. Główną różnicą jest więc sposób reprezentacji i przetwarzania informacji - baza wiedzy skupia się na logice i wnioskowaniu, podczas gdy baza danych koncentruje się na efektywnym przechowywaniu i dostępie do danych.

Zapytania

Ich logika polega na pobieraniu informacji z zapewnionej bazy wiedzy zazwyczaj w tym samym pliku typu Prolog i zwracaniu ekspertowych informacji. W projekcie utworzyliśmy dwa zapytania aby zapewnić wyszukanie wyników po 4 kategoriach:

zapytanieDopasowanieAnime

Opisane zapytanie w Prologu służy do znajdowania dopasowanych anime na podstawie określonych filtrów. Funkcja `znajdzDopasowaneAnime` ma sześć parametrów wejściowych: Tytuł, Gatunki, Odcinki, Ocena, ID, Format i Obejrzenia. Używamy go przy wykorzystaniu gatunków gdyż sposób napisanie owego zapytania pozwala na wyszukanie po jednym jak i wielu wynikach gatunków

Zapytanie składa się z kilku warunków, które są sprawdzane dla każdego faktora anime. Warunki te są opcjonalne i mogą być pominięte, jeśli odpowiadające im argumenty są niewiążące (var oznacza niewiązącą zmienną). Jeśli argumenty nie są niewiążące, stosowane są filtry na podstawie tych argumentów.

Na przykład, jeśli Tytuł nie jest niewiążący, stosowany jest filtr `zastosujFiltrTytułu(ID, Tytuł)`, który sprawdza, czy fakt anime ma pasujący tytuł. Analogicznie, jeśli Gatunki nie jest niewiążący, stosowany jest filtr `zastosujFiltrGatunków(ID, Gatunki, AnimeGatunki)`, który sprawdza, czy fakt anime ma wszystkie gatunki z listy Gatunki. Pozostałe filtry działają analogicznie dla pozostałych argumentów.

Funkcja pomocnicza `containsAllGenres` jest używana przez filtr `zastosujFiltrGatunków` i sprawdza, czy lista Gatunki zawiera wszystkie gatunki z listy AnimeGatunki.

W rezultacie, wykonanie zapytania `znajdzDopasowaneAnime` z określonymi filtrami zwróci listę anime, które spełniają te warunki, wraz z odpowiadającymi im wartościami Tytuł, Gatunki, Odcinki, Ocena, ID, Format i Obejrzenia.

Zapytanie jako kod w pliku:

```
znajdzDopasowaneAnime(Tytuł, Gatunki, Odcinki, Ocena, ID, Format, Obejrzenia) :-
    anime(ID, Tytuł, AnimeGatunki, Format, Odcinki, Ocena, Obejrzenia),
    (var(Tytuł) ; zastosujFiltrTytułu(ID, Tytuł)),
    (var(Gatunki) ; zastosujFiltrGatunków(ID, Gatunki, AnimeGatunki)),
    (var(Odcinki) ; zastosujFiltrOdcinków(ID, Odcinki)),
    (var(Ocena) ; zastosujFiltrOceny(ID, Ocena)),
    (var(Format) ; zastosujFiltrFormatu(ID, Format)),
    (var(Obejrzenia) ; zastosujFiltrObejrzeń(ID, Obejrzenia)).

zastosujFiltrTytułu(ID, Tytuł) :-
    anime(ID, Tytuł, _, _, _, _, _).

zastosujFiltrGatunków(ID, Gatunki, AnimeGatunki) :-
    anime(ID, _, AnimeGatunki, _, _, _, _),
    containsAllGenres(Gatunki, AnimeGatunki).

zastosujFiltrOdcinków(ID, Odcinki) :-
    anime(ID, _, _, _, Odcinki, _, _).

zastosujFiltrOceny(ID, Ocena) :-
    anime(ID, _, _, _, _, Ocena, _).

zastosujFiltrFormatu(ID, Format) :-
    anime(ID, _, _, Format, _, _, _).

zastosujFiltrObejrzeń(ID, Obejrzenia) :-
    anime(ID, _, _, _, _, _, Obejrzenia).

containsAllGenres([], _).
containsAllGenres([Gatunek|Reszta], AnimeGatunki) :-
    member(Gatunek, AnimeGatunki),
    containsAllGenres(Reszta, AnimeGatunki).
```


Przykładowy wynik zapytania z wykorzystaniem narzędzia SWI-PROLOG

W tym przypadku, zapytanie ma zwrócić anime o gatunkach "School" i "Action", ocenie wyższej niż 8 oraz formacie "TV". Odpowiedzią na to zapytanie są dwie opcje:

```
?- znajdzDopasowaneAnime(Tytuł, ['School','Action'], Odcinki, Ocena, ID, Format, Obejrzenia),Ocena>8,Format='TV'.
Tytuł = 'Code Geass: Hangyaku no Lelouch',
Odcinki = 25,
Ocena = 8.83,
ID = 1575,
Format = 'TV',
Obejrzenia = 715151 ;
Tytuł = 'Kill la Kill',
Odcinki = 24,
Ocena = 8.23,
ID = 18679,
Format = 'TV',
Obejrzenia = 508118 ;
false.
```

zapytanieAnime

To zapytanie jest wykorzystywane w każdej sytuacji nie wymagającej wyszukania przez Gatunki gdyż gatunki są jako tablica. Natomiast lepiej ogólnie przeszukuje bazę wiedzy.

```
znajdzAnime(Tytuł, Gatunki, Odcinki, Ocena, ID, Format, Obejrzenia) :-
    anime(ID, Tytuł, Gatunki, Format, Odcinki, Ocena, Obejrzenia),
    (var(Tytuł) ; anime(ID, Tytuł, _, _, _, _, _)),
    (var(Gatunki) ; anime(ID, _, Gatunki, _, _, _, _)),
    (var(Odcinki) ; anime(ID, _, _, Odcinki, _, _)),
    (var(Ocena) ; anime(ID, _, _, _, Ocena, _)),
    (var(Format) ; anime(ID, _, _, Format, _, _)),
    (var(Obejrzenia) ; anime(ID, _, _, _, _, Obejrzenia)).
```

Dodatkowe warunki w zapytaniach

W języku Prolog istnieje możliwość dodania dodatkowych warunków do zapytań w celu zawężenia wyników. Poprzez dodanie warunków takich jak 'ocena > 8', można wybrać tylko te fakty, gdzie wartość pola 'ocena' jest większa niż 8. To umożliwia użytkownikom dokładne określenie kryteriów wyszukiwania i otrzymywanie bardziej precyzyjnych wyników."

Ten sposób jest bardzo często używany w naszym projekcie eksperta, poniżej przykład.

```
?- znajdzDopasowaneAnime(Tytuł, ['School','Action'], Odcinki, Ocena, ID, Format, Obejrzenia),Ocena>8,Format='TV'.
```

4. Opis aplikacji Java Spring

Serwis -> Klasa ta odpowiada za komunikację z prologiem, posiada dwie metody wysyłające zapytania, jedna odpowiada za nawiązanie połączenia a druga za wysłanie zapytania po nawiązanym połączeniu o wynik zebranych danych od użytkownika

Metoda łącząca z plikiem prolog zawierającym bazę i zapytania

- Inicjalizacja biblioteki JPL: Na początku programu, biblioteka JPL (Java Prolog Bridge) jest przygotowywana do użycia. Jest to narzędzie, które umożliwia komunikację między językiem Java a językiem Prolog.
- Konsultowanie pliku Prologu: Następnie, program otwiera plik Prologu o nazwie "baza.pl" i sprawdza jego zawartość. Polecenie `consult('baza.pl')` oznacza, że program chce skonsultować ten plik, czyli przetworzyć jego treść i przygotować go do dalszego użycia.
- Sprawdzanie rezultatu zapytania: Po skonsultowaniu pliku, program sprawdza, czy operacja się powiodła. Wywołanie metody `hasSolution()` na obiekcie `Query` sprawdza, czy udało się znaleźć rozwiązanie. Jeśli tak, to oznacza, że plik Prologu został załadowany poprawnie. W przeciwnym razie, program wyświetla komunikat o błędzie, informujący o niepowodzeniu ładowania pliku.

```
@Service
public class AnimeService {

    public static void getConnectionWithPrologFile(){
        String[] initArgs = {"-q"};
        JPL.init(initArgs);

        Query query = new Query( s: "consult('baza.pl')");
        if (query.hasSolution()) {
            System.out.println("Plik Prologu załadowany pomyślnie.");
        } else {
            System.out.println("Błąd podczas ładowania pliku Prologu.");
        }
    }
}
```

Metoda wysyłająca zapytanie o wyniki z bazy

Wynikiem poniższej metody jest lista wyników z bazy. Przyjmuje ona cztery argumenty które są zebranymi informacjami od użytkownika zastosowanymi w zapytaniu o wyniki. Query jest klasą zapewnioną przez bibliotekę jpl7. Widzimy poniżej warunek if który stosuje jedną z metod wspomnianej klasy Query i sprawdza czy wogóle otrzymaliśmy jakieś wyniki, metoda hasNext pozwala nam przejść do następnego wyniku następnej pozycji która pasuje a cała pętla przekłada zebrane wyniki z bazy do listy zwracanej return.

```
public static List<Map<String, Term>> getSolutionsFromQueryWithAnime(String zapytanie, String epizody, String oceny, String format){
    Query goQuery = new Query(s: zapytanie + epizody+ oceny + format+ ".");
    List<Map<String, Term>> solutions = new ArrayList<>();

    if (goQuery.hasSolution()) {
        while (goQuery.hasNext()) {
            Map<String, Term> solution = goQuery.nextSolution();
            solutions.add(solution);
        }
    }
    return solutions;
}
```

Encja -> Klasa która imituje fakty z bazy wiedzy posiada ona zmienne będące kategorią po których wyszukuje wyniki. Używamy ją w klasie o której później wspomnę będące kontrolerem aby przechowywać pobierane dane poprzez utworzenie z niej obiektu podczas startu programu. Poniżej zmiennymi mamy konstruktor deklarujący sposób tworzenia nowego obiektu anime gettery oraz settery, metody służące do zapisywania i pobierania informacji z obiektu oraz toString() aby wyświetlić wizualnie poprawnie owe zmienne.

Poniżej fragment owej klasy bez pełnych getterów i setterów.

```
public class Anime {

    private String genres;
    private String format;
    private String episodes;
    private String rating;

    public Anime() {
    }

    public Anime(String genres, String format, String episodes, String rating) {
        this.genres = genres;
        this.format = format;
        this.episodes = episodes;
        this.rating = rating;
    }

    public Anime(String format, String episodes, String rating) {
        this.format = format;
        this.episodes = episodes;
        this.rating = rating;
    }

    @Override
    public String toString() {
        return "Anime{" +
            "genres=" + genres +
            ", format='" + format + '\'' +
            ", episodes=" + episodes +
            ", rating=" + rating +
            '}';
    }

    public String getGenres() { return genres; }

    public void setGenres(String genres) { this.genres = genres; }
```

Kontroler -> Klasa zawierająca end-pointy czyli deklarująca ostateczne warunki sposób korzystania z WEB API. Metody tutaj zawarte mają zaimplementowane logiki i wykorzystanie encji i serwisu. Na początku klasy deklaruje obiekty klasy Anime aby móc go użyć w każdej metodzie oraz zmienną string przechowującą zapytanie przekazywane później do serwisu i zmieniane pod wpływem danych od użytkownika. Adnotacja @Controller oznacza dla projektu klasę odpowiedzialną za przetwarzanie żądań HTTP oraz zarządzanie logiką biznesową natomiast @RequestMapping("/api") pod jakim url będą metody.

```
@Controller
@RequestMapping("/api")
public class AnimeController {

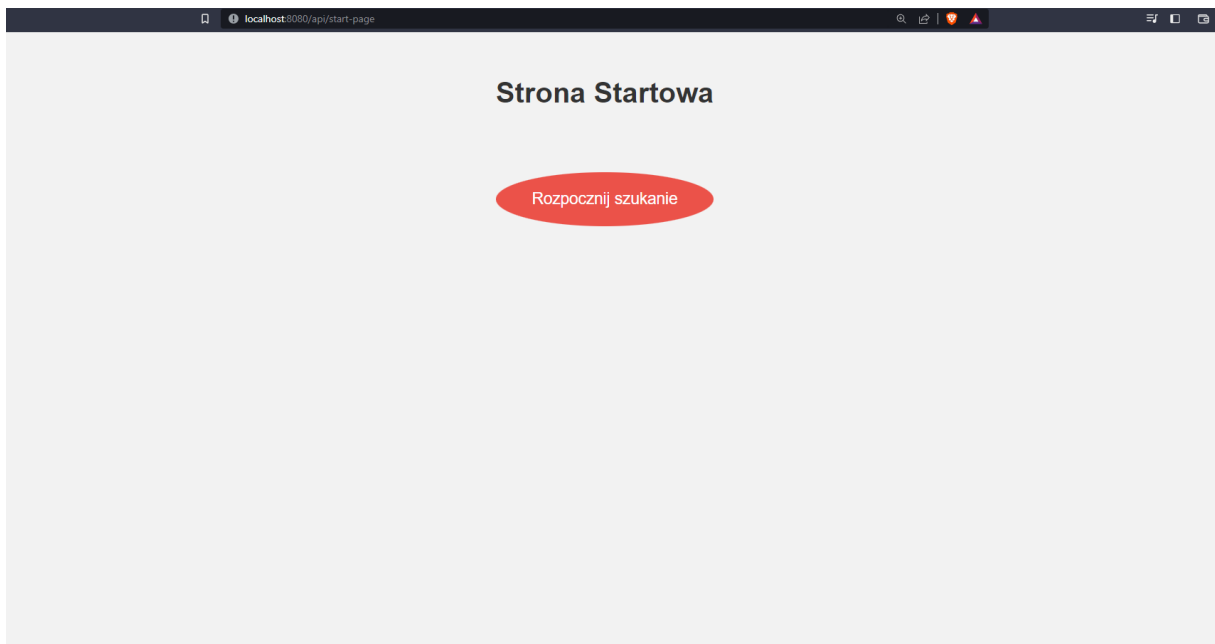
    Anime animeQueryKategorie ;
    String zapytanie;
```

Metoda ze stroną startową

Występuje tutaj kolejna adnotacja określająca że strona startowa będzie pod url /api/start-page. Odpowiada ona głównie za zerowanie zebranych informacji po wyświetlonych już wynikach jak i przygotowanie przy starcie od początku. Zwracanym obiektem jest plik html który zawiera sposób wyświetlenia strony. Jest to możliwe dzięki wykorzystaniu wspomnianego przy na początku silnika Thymeleaf dodatkowo aby było to możliwe jest potrzebna wcześniej dodana adnotacja @Controller.

```
//STRONA STARTOWA
@GetMapping("/start-page")
public String startPage() {
    zapytanie = "znajdzAnime(Tytuł, Gatunki, Odcinki, Ocena, ID, Format, Obejrzenia)";
    animeQueryKategorie = new Anime( genres: "Gatunki", format: " ", episodes: " ", rating: " ");
    return "startpage";
}
```

Wynik metody zwracany html



Metody pobierające informacje

Pobieram łącznie od użytkownika kryteria według 4 kategorii i dla każdej jest jedna strona

- Get określa że pobieram informacje czyli w tym przypadku wygląd strony i zachodzi od razu po przejściu na url. <http://localhost:8080/api/format>
- Natomiast Post zachodzi po przesłaniu danych, jako argument mam string który jest przekazywane z pliku html to jest również zasługa Thymeleaf. Post zachodzi tylko po kliknięciu przycisku z wystaniem wprowadzonego kryterium i jest to zdefiniowanej w pliku html o którym wspomnę dalej. Warunek if sprawdza czy otrzymywałem jakieś dane i wtedy podmienia podstawowy zapis aby wyszukać wszystko tej kategorii na kryteria użytkownika. Metoda POST zwraca przekierowanie na następną stronę w tym wypadku pobieranie ocen anime. Ostatnia strona z kryteriami pobiera ilość odcinków i po niej akurat występuje już przejście do strony z wynikami

```
//FORMAT
@GetMapping("/format")
public String expertPageFormat(Model model) { return "expertpageformat"; }
@PostMapping("/format")
public String processAnswerFormat(@RequestParam("answer") String answer) {
    if (!answer.isEmpty()) {animeQueryKategorie.setFormat(",Format =" +answer);}
    return "redirect:/api/oceny";
}
```

Wygląd strony

localhost:8080/api/format

Anime Expert

Jakie format cię interesuje?

Strona startowa

Przykładowe odpowiedzi to 'TV', 'Movie', 'OVA' lub 'Special'

Przejdź dalej

Metoda z wynikami

Argument klasy Model jest wykorzystywany przez Thymeleaf do przekazywania informacji między java a plikiem html dzięki czemu przekazuję wyniki otrzymane z Prologa i wyświetlam je w oprawie graficznej. Widać tutaj wykorzystanie metod z klasy Serwis czyli połączenie z bazą wiedzy i zapytaniem w prologu i kolejna metoda która przekazuje kryteria i odbiera listę. Odebrana lista za pomocą metody addAttribute przekazuje wyniki pod postacią listy do pliku html resultpage.

```
//WYNIKI
@GetMapping("/results")
public String resultsPage(Model model) {

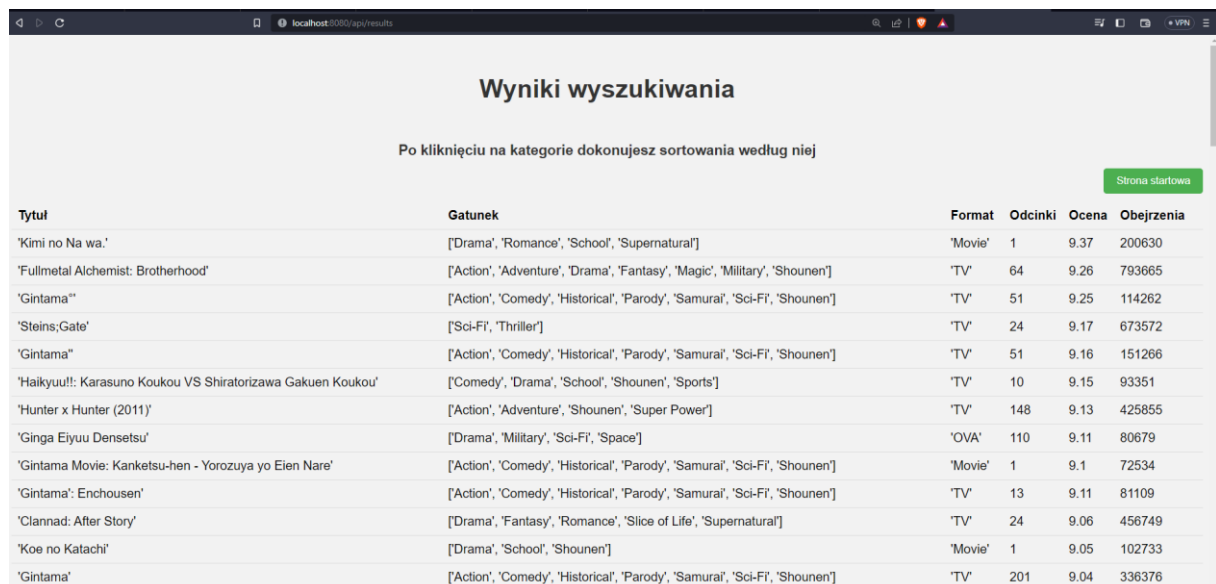
    AnimeService.getConnectionWithPrologFile();

    List<Map<String, Term>> solutions = AnimeService.getSolutionsFromQueryWithAnime(
        zapytanie ,
        animeQueryKategorie.getEpisodes(),
        animeQueryKategorie.getRating(),
        animeQueryKategorie.getFormat());

    model.addAttribute( attributeName: "results", solutions);

    return "resultspage";
}
```

Wygląd wizualny strony wynikowej



Wyniki wyszukiwania						
Po kliknięciu na kategorie dokonujesz sortowania według niej						
Strona startowa						
Tytuł	Gatunek	Format	Odcinki	Ocena	Obejrzenia	
'Kimi no Na wa.'	[Drama, 'Romance', 'School', 'Supernatural']	'Movie'	1	9.37	200630	
'Fullmetal Alchemist: Brotherhood'	[Action, 'Adventure', 'Drama', 'Fantasy', 'Magic', 'Military', 'Shounen']	'TV'	64	9.26	793665	
'Gintama'	[Action, 'Comedy', 'Historical', 'Parody', 'Samurai', 'Sci-Fi', 'Shounen']	'TV'	51	9.25	114262	
'Steins;Gate'	[Sci-Fi, 'Thriller']	'TV'	24	9.17	673572	
'Gintama'	[Action, 'Comedy', 'Historical', 'Parody', 'Samurai', 'Sci-Fi', 'Shounen']	'TV'	51	9.16	151266	
'Haikyuu!!: Karasuno Koukou VS Shiratorizawa Gakuen Koukou'	[Comedy, 'Drama', 'School', 'Shounen', 'Sports']	'TV'	10	9.15	93351	
'Hunter x Hunter (2011)'	[Action, 'Adventure', 'Shounen', 'Super Power']	'TV'	148	9.13	425855	
'Ginga Eiyuu Densetsu'	[Drama, 'Military', 'Sci-Fi', 'Space']	'OVA'	110	9.11	80679	
'Gintama Movie: Kanketsu-hen - Yorozuya yo Eien Nare'	[Action, 'Comedy', 'Historical', 'Parody', 'Samurai', 'Sci-Fi', 'Shounen']	'Movie'	1	9.1	72534	
'Gintama: Enchousen'	[Action, 'Comedy', 'Historical', 'Parody', 'Samurai', 'Sci-Fi', 'Shounen']	'TV'	13	9.11	81109	
'Clannad: After Story'	[Drama, 'Fantasy', 'Romance', 'Slice of Life', 'Supernatural']	'TV'	24	9.06	456749	
'Koe no Katachi'	[Drama, 'School', 'Shounen']	'Movie'	1	9.05	102733	
'Gintama'	[Action, 'Comedy', 'Historical', 'Parody', 'Samurai', 'Sci-Fi', 'Shounen']	'TV'	201	9.04	336376	

Plik maven konfiguracyjny

Jest to plik od razu zapewniony przy tworzeniu nowego projektu z wykorzystaniem narzędzia Spring. Umieszczane w nim są wszystkie biblioteki które zapewniają nam metody i klasy potrzebne w projekcie przykładowo chcemy użyć JPL wersji 7 do prologa dodajemy bibliotekę znaną na stronie oficjalnej maven i wklejamy do pliku pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
  </dependency>
  <!-- https://mvnrepository.com/artifact/jpl/jpl -->
  <dependency>
    <groupId>jpl</groupId>
    <artifactId>jpl</artifactId>
    <version>7.4.0</version>
  </dependency>
```

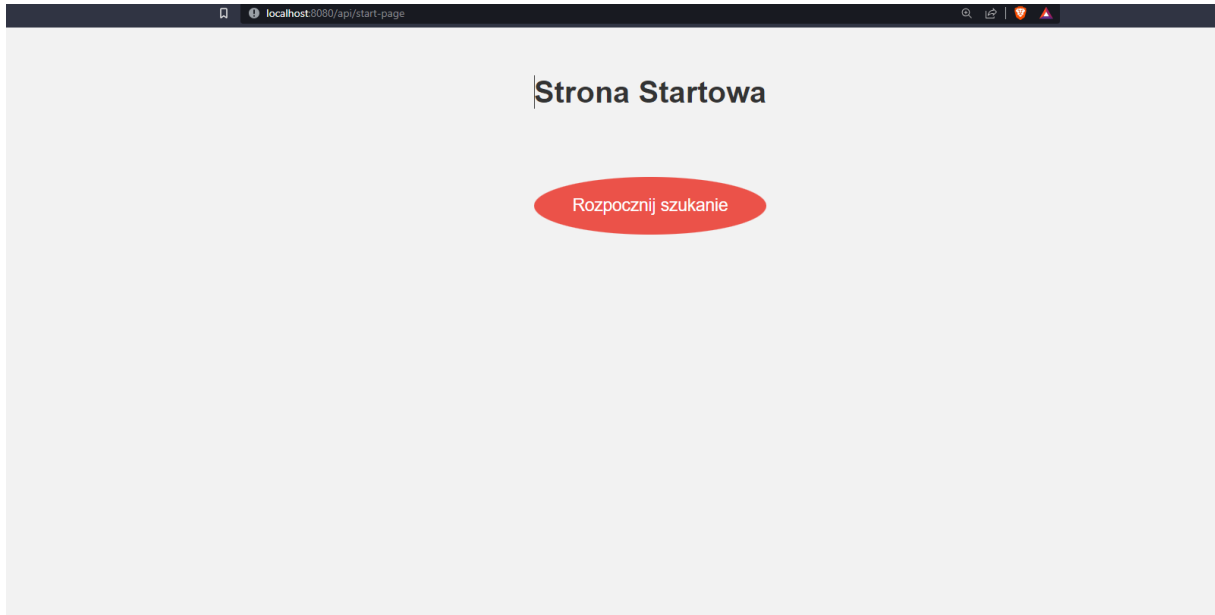
Pliki html i css

Zawierają deklaracje w jaki sposób wizualnie wyświetlamy przedstawione w metodzie metody oraz posiadają kod który na przykład zablokowuje przejście do następnej strony bez wprowadzenia poprawnych danych. W stronie związanej z wynikami są zawarte jeszcze elementy JavaScript które pomagają w sortowaniu wyników, użytkownik może kliknąć na podaną kategorię i posortować rosnąco lub malejąco według niej.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Anime expert</title>
  <link rel="stylesheet" type="text/css" th:href="@{/css/expertstyle.css}">
</head>
<body>
  <h1>Anime Expert</h1>
  <h1>Jakie format cię interesuje?</h1>
  <div class="button-container">
    <button onclick="window.location.href='start-page'">Strona startowa</button>
  </div>
  <div class="expert-container">
    <div class="text-container">
      <form action="/api/format" method="POST">
        <textarea id="message" name="answer" rows="10"></textarea>
        <p>Przykładowe odpowiedzi to 'TV' , 'Movie' , 'OVA' lub 'Special'</p>
        <button class="send-button" type="submit">Przejdź dalej</button>
      </form>
    </div>
  </div>
</body>
</html>
```

5. Scenariusz użycia

Kroko 1. Jeśli projekt jest włączony użytkownik przechodzi na stronę startową <http://localhost:8080/api/start-page> gdzie po kliknięciu przycisku przechodzi do pierwszego zapytania o kryterium



Kroko 2. Użytkownik wprowadza gatunki którego go interesują według wytycznych, może również nic nie wpisać i kliknąć przejdź dalej nie deklarując preferencji zostanie to uwzględnione w wynikach. Może na każdym etapie wprowadzania kryteriów cofnąć się i rozpocząć test od nowa. Przycisk dalej pozwala przejść do następnych kryteriów.



Krok 3. Użytkownik wprowadza format według zaleceń oczywiście może zostawić to pole jako puste

The screenshot shows a web browser at the URL `localhost:8080/api/format`. The page has a light gray background and a dark header bar. The main heading is "Anime Expert" in bold black text. Below it is the question "Jakie format cię interesuje?". A large white text input field contains the text "'TV'". Below the input field, there is a line of text: "Przykładowe odpowiedzi to 'TV' , 'Movie' , 'OVA' lub 'Special'". At the bottom of the form is a green button with the text "Przejdź dalej".

Krok 4. Strona z ocenami w której możemy wpisać znak porównania i liczbę nawet jako ułamkową w celu znalezienia wyników z podanymi ocenami np. "< 7". Jeśli wprowadzimy nie pożądane znaki przez system wyświetli się komunikat z informacją i dopóki nie pozostanie puste okienko nic nie deklarując lub poprawnie wprowadzone dane przycisk przejścia dalej nie działa.

The screenshot shows a web browser at the URL `localhost:8080/api/oceny`. The page has a light gray background and a dark header bar. The main heading is "Anime Expert" in bold black text. Below it is the question "Jakie średnie oceny na popularnych stronach oczekujesz?". In the top right corner, there is a green button with the text "Strona startowa". A large white text input field contains the text "> 8". Below the input field is a green button with the text "Przejdź dalej". At the bottom right of the page, there is a small gray text link that says "Aktywuj system Windows".

Krok 5. Po przejściu do ostatniego kryterium wygląda to tak samo jak z oceną. Tym razem po kliknięciu przejdź dalej otrzymamy wyniki

localhost:8080/api/odcinki

Anime Expert

Ilość odcinków jaką oczekujesz?

> 12

Wprowadź znak >, < lub =, a następnie liczbę.

Przejdź dalej

Krok 6. Strona z wynikami naszych wytycznych. Klikając na pogrubione nazwy kategorii możemy je sortować rosnąco po jednym kliknięciu i malejąco o ponownym. Klikając przycisk strona startowa wracamy do kroku 1 i tracimy otrzymane wyniki które się resetują.

localhost:8080/api/results

Wyniki wyszukiwania

Po kliknięciu na kategorie dokonujesz sortowania według niej

Strona startowa

Tytuł	Gatunek	Format	Odcinki	Ocena	Obejrzenia
'Fullmetal Alchemist: Brotherhood'	[Action, Adventure, Drama, Fantasy, Magic, Military, Shounen]	TV	64	9.26	793665
'Gintama'	[Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen]	TV	51	9.25	114262
'Steins;Gate'	[Sci-Fi, Thriller]	TV	24	9.17	673572
'Gintama'	[Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen]	TV	51	9.16	151266
'Hunter x Hunter (2011)'	[Action, Adventure, Shounen, Super Power]	TV	148	9.13	425855
'Gintama: Enchousen'	[Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen]	TV	13	9.11	81109
'Clannad: After Story'	[Drama, Fantasy, Romance, Slice of Life, Supernatural]	TV	24	9.06	456749
'Gintama'	[Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen]	TV	201	9.04	336376
'Code Geass: Hangyaku no Lelouch R2'	[Action, Drama, Mecha, Military, Sci-Fi, Super Power]	TV	25	8.98	572888
'Haikyuu!! Second Season'	[Comedy, Drama, School, Shounen, Sports]	TV	25	8.93	179342
'Shigatsu wa Kimi no Uso'	[Drama, Music, Romance, School, Shounen]	TV	22	8.92	416397
'Code Geass: Hangyaku no Lelouch'	[Action, Mecha, Military, School, Sci-Fi, Super Power]	TV	25	8.83	715151
'Haikyo no Ippo'	[Comedy, Drama, Shounen, Sports]	TV	75	8.83	157670

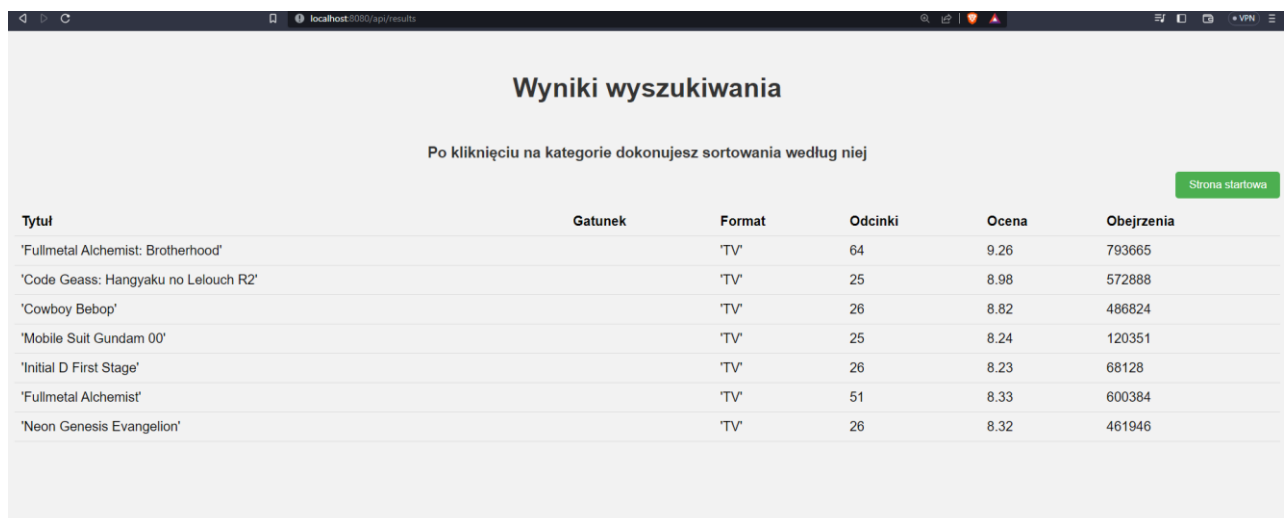
6. Zalecenia przy korzystaniu z projektu

- Projekt został zrealizowany z wykorzystaniem środowiska programistycznego Intelij które zalecamy do wykorzystania, nie posiadamy pewności czy zadziała na innych. Mogą wystąpić wtedy problemy z prologiem
- Wymagane jest poprawne zainstalowanie środowiska pod java czyli pobranie JDK oraz języka Prolog wraz z pobraniem narzędzia SWI-PROLOG który zapewnia JPL bibliotekę w pakiecie.
- Prolog jest starszym językiem mimo ciągłego wspierania go wszelakimi bibliotekami wymagane będzie dodanie w systemie ścieżek naprowadzających na pliki jar zapewniające JPL dla połączenia z java.
- Plik prolog jest zwarty w projekcie interfejsu od razu

7. Przykład wyniku

Wynik dla wyszukania po:

- Gatunek: Drama i akcja
- Format: TV
- Oceny : powyżej średniej 8/10
- Odcinki : powyżej 15



Wyniki wyszukiwania					
Po kliknięciu na kategorie dokonujesz sortowania według niej					
Strona startowa					
Tytuł	Gatunek	Format	Odcinki	Ocena	Obejrzenia
'Fullmetal Alchemist: Brotherhood'		TV	64	9.26	793665
'Code Geass: Hangyaku no Lelouch R2'		TV	25	8.98	572888
'Cowboy Bebop'		TV	26	8.82	486824
'Mobile Suit Gundam 00'		TV	25	8.24	120351
'Initial D First Stage'		TV	26	8.23	68128
'Fullmetal Alchemist'		TV	51	8.33	600384
'Neon Genesis Evangelion'		TV	26	8.32	461946

8. Podsumowanie

Ten projekt jest aplikacją webową, która służy do doradzania użytkownikom w wyborze anime na podstawie ich preferencji. Użytkownik ma możliwość wprowadzenia kryteriów takich jak gatunki, liczba odcinków, ocena i format, a następnie system proponuje mu listę anime spełniających te kryteria. Aplikacja korzysta z języka Prolog oraz bazy wiedzy, która zawiera informacje o różnych anime. Wykorzystuje również framework Spring oraz technologię Java do implementacji logiki biznesowej i komunikacji z bazą danych. Dzięki temu projektowi użytkownicy mogą otrzymać spersonalizowane rekomendacje anime, co pomaga im w podjęciu decyzji dotyczących oglądania nowych tytułów. Mimo to nie jest to idealna wersja do publikacji i użytku zewnętrznego. Wymaga usprawnienia możliwości wprowadzania danych gdyż jako użytkownik jest to dość irytujące wprowadzając ciągle cudzysłowia lub nawiasy oraz zabezpieczenia przed nie pożądanymi danymi [przy wyszukiwaniu].

9. Bibliografia

- Dokumentacja dla języka prolog i narzędzia SWI-PROLOG https://www.swi-prolog.org/pldoc/doc_for?object=manual
- Dokumentacja biblioteki JPL <https://jpl7.org/>
- Zrozumienie działania języka prolog <https://www.tutorialspoint.com/prolog/index.htm>
- Aspekt teoretyczny wikipedia [https://pl.wikipedia.org/wiki/Prolog_\(j%C4%99zyk_programowania\)](https://pl.wikipedia.org/wiki/Prolog_(j%C4%99zyk_programowania))

10. Autorzy

- Mateusz Rysiewski Informatyka stosowana gr.1 nr.118850

