**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ІВАНА ФРАНКА**

Факультет прикладної математики та інформатики

(повне найменування назва факультету)

Кафедра дискретного аналізу та інформаційних систем

(повна назва кафедри)

**КУРСОВА РОБОТА**

на тему:

Розробка web-орієнтованої системи для централізованої бази даних фільмів та постерів українською мовою

Студента 3 курсу, групи ПМІ-33 ,

напряму підготовки інформатика

Романюка Б.І.

(прізвище та ініціали)

Керівник ас. Квасниця Г.А.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Львів – 2018

**Зміст**

[Вступ 3](#_Toc515568007)

[1. Постановка задачі 4](#_Toc515568008)

[2. Побудова парсеру та парсинг інформації 6](#_Toc515568009)

[2.1 Побудова графічного інтерфейсу парсеру 7](#_Toc515568010)

[2.2 Логінування програми під виглядом звичайного користувача 9](#_Toc515568011)

[2.3. Зберігання cookie файлів та здійснення GET запитів 10](#_Toc515568012)

[2.4 Парсинг фільмів 12](#_Toc515568013)

[2.4.1 Перший етап розпізнавання фільмів 12](#_Toc515568014)

[2.4.2 Другий етап розпізнавання фільмів 15](#_Toc515568015)

[2.5 Розпізнавання жанру та країни виробництва для окремого фільму 20](#_Toc515568016)

[2.6 Зберігання інформації в бінарному форматі 21](#_Toc515568017)

[2.7 Завантаження постерів 23](#_Toc515568018)

[2.8 Оновлення бази фільмів 24](#_Toc515568019)

[2.9 Пошук за різними критеріями 26](#_Toc515568020)

[2.10 Розпізнавання та зберігання всіх фільмів та постерів. 27](#_Toc515568021)

[3. Побудова web-орієнтованої системи 29](#_Toc515568022)

[3.1 Модель бази даних 29](#_Toc515568023)

[3.2 ER-діаграма бази даних 30](#_Toc515568024)

[3.3 Заповнення бази даних фільмами 31](#_Toc515568025)

[3.4 Графічний інтерфейс 32](#_Toc515568026)

[3.5 Виведення даних 33](#_Toc515568027)

[3.6 Функції пошуку 35](#_Toc515568028)

[3.7 Реєстрація користувачів та вхід 36](#_Toc515568029)

[3.8 Відгуки до фільмів 37](#_Toc515568030)

[Висновки 38](#_Toc515568031)

[Список використаних джерел 39](#_Toc515568032)

# Вступ

На сьогоднішній день іноземні та українські аналоги не мають функціоналу для розширеного пошуку фільмів за різними властивостями та їхнього обговорення користувачами, переважно такі сайти мають лише коротку інформацію про фільм, акторський склад та сюжет.

Розглянемо кілька прикладів:

1. Англійський сайт **imdb.com** (Internet Movie DataBase) найпопулярніший сайт з пошуку фільмів у світі, проте хоч він і містить локалізацію 5 мовами, серед них немає української, також сайт не дає вичерпної інформації про сюжет фільму, обмежуючись лише одним реченням. До плюсів сайту можна віднести наявність та можливість пошуку за певними критеріями та рейтингову систему, в якій можуть брати участь усі зареєстровані користувачі, проте реєстрація не дає змоги обговорювати фільми, так як відгуки до фільмів можуть залишати лише кінокритики.
2. Український сайт **kino-teatr.ua** один з найпопулярніших в Україні сайтів з пошуку фільмів, проте він має теж свої недоліки. По-перше увесь сайт не ведеться українською мовою, а частково російською. По-друге на сайті присутня досить мала кількість фільмів, навіть дуже відомих. До плюсів сайту можна віднести можливість рейтингувати фільм, будучи не зареєстрованим користувачем, а також можливість залишати відгуки зареєстрованим користувачам.

Український сайт **hurtom.com** є найбільш привабливим у плані обговорення фільмів та їх оцінювання. Проте хоч вся інформація і зберігається українською мовою, але вона зберігається в окремому html блоці, що унеможливлює пошук за жанром, роком і тд. До великих плюсів сайту можна віднести існування дочірнього сайту posters.hurtom.com, який містить близько 6 тисячі українізованих постерів.

Тому виникла ідея створити централізовану базу даних фільмів українською мовою, в якій українці мали б змогу шукати та обговорювати фільми.

# Постановка задачі

Метою даної курсової роботи є створення web-орієнтованої системи, яка б містила велику кількість фільмів та інформації про них і вся інформація була б при цьому українською мовою. Для створення самого сайту буде використано технологію ASP .NET Core з використанням патерну проектування MVC. Сайт буде давати можливість користувачам реєструватися, логінуватися, шукати, обговорювати фільми та багато іншого функціоналу і все це виключно українською мовою. Інформація про всі фільми буде зберігатися у базі даних, в якій таблиця фільмів міститиме всі колонки з властивостями за якими можна здійснювати пошук(назва, рік, жанр, країна, режисер, актори, сюжет, тривалість, а також українізований постер(у більшості випадків, в інших оригінальний)). Також кожен фільм міститиме рейтинг найпопулярнішого сайту з пошуку фільмів imdb.com. Основну увагу сайту буде звернено саме на можливість поглибленого пошуку використовуючи такі функції як Full Text Search і Indexes, що дасть змогу ефективно і навіть не знаючи всієї інформації шукати потрібний фільм, оскільки Full Text Search дає змогу здійснювати пошук за ключовими словами в будь якому порядку. Користувачі зможуть також залишати відгуки до фільмів.

Заповнювати сайт фільмами буде спеціально побудований парсер, який за допомогою регулярних виразів(Regular Expressions) та парсингу стрінгів наповнить сайт тисячами фільмів, інформацією про них та українськими постерами. Більша частина інформації буде братися з сайтів hurtom.com та posters.hurtom.com, оскільки вони уже містять необхідну інформацію українською мовою та постери, залишається лише її розпізнати та зберегти. Також частина інформації буде братися з сайту imdb.com та omdbapi.com. Парсер буде побудовано з використанням технології WPF та патерну MVVM. Оскільки сайт hurtom.com вимагає у значній частині випадків реєстрацію, то для здійснення GET запитів буде створено спеціальний клас, який буде зберігати Cookie файли і давати змогу парсеру працювати під виглядом зареєстрованого користувача. Для парсингу самої HTML структури сторінок з фільмами одних регулярних виразів і функцій бібліотечного класу String буде не достатньо, тому буде також використано сторонню бібліотеку HtmlAgilityPack, яка дає зручну функціональність для парсингу DOM структури сторінок. Окрім цього програма міститиме функціональність для зберігання постерів та доповнення уже існуючої бази даних.

До особливих функцій сайту можна віднести функцію розширеного пошуку, яка буде давати наступні можливості:

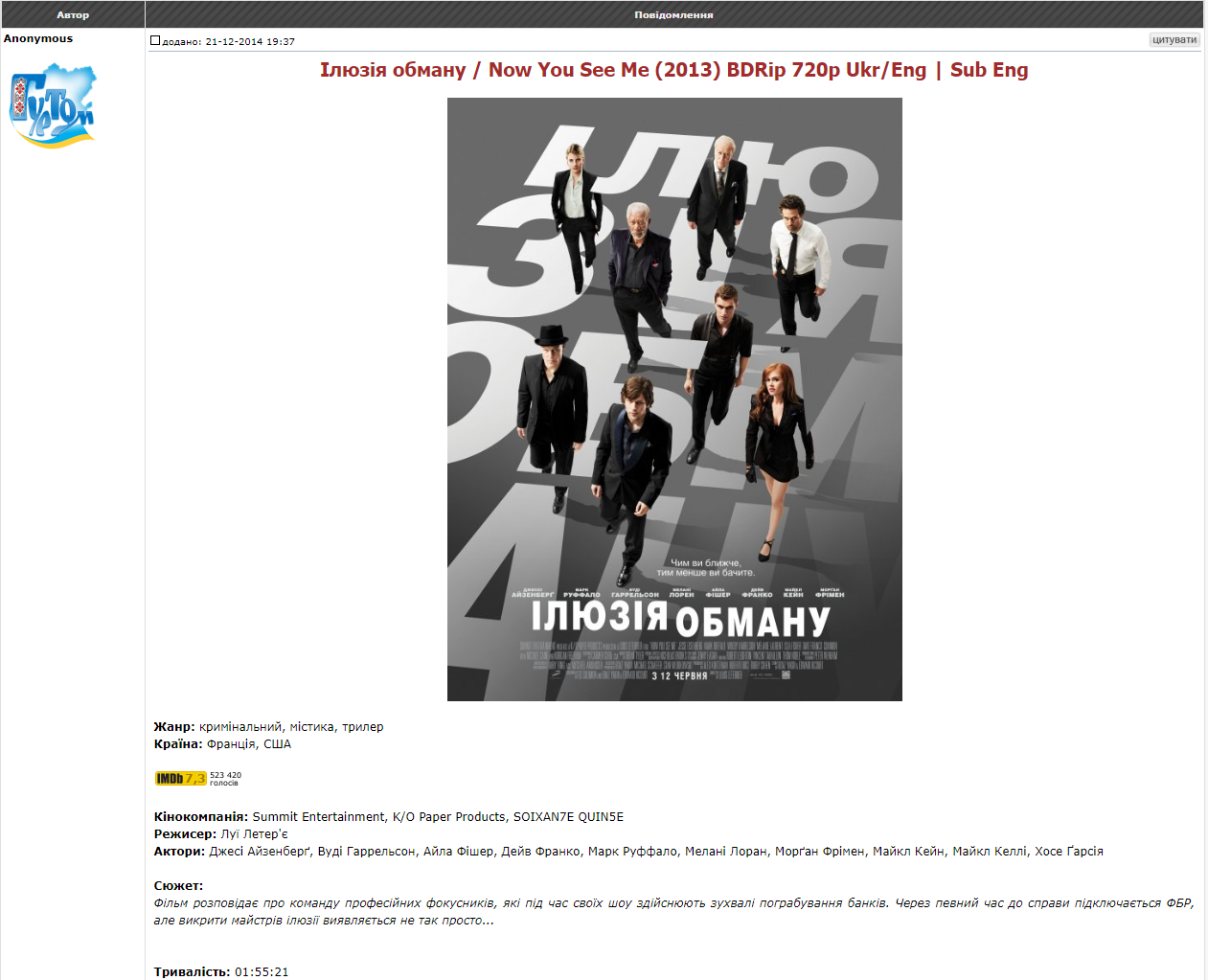
-пошук за назвою, як українською так і оригінальною;  
-фільтрування за жанрами;

-фільтрування за роками;

-фільтрування за країною випуску.

# Побудова парсеру та парсинг інформації

Як зазначалося раніше основна частина інформації про фільми буде братися з сайту hurtom.com, цей сайт має структуру форуму, де кожен фільм зберігається в окремій темі у першому повідомленні, як це показано на ***Рисунку 2.1***. Також варто зауважити, що всі фільми зберігаються в окремому розділі, поділеному на сторінки по 45 фільмів на сторінку.



***Рисунок 2.1.*** Структура опису фільму на сайті hurtom.com

Можна зауважити, що ключові слова, за якими в подальшому буде здійнюватися пошук виділені жирним шрифтом, та завершуються двокрапкою, також дане представлення містить українізований постер та картинку з рейтингом найпопулярнішого сайту з пошуку фільмів imdb.com, при кліканні на яку користувача переадресує на відповідну сторінку з фільмом з сайту imdb.com

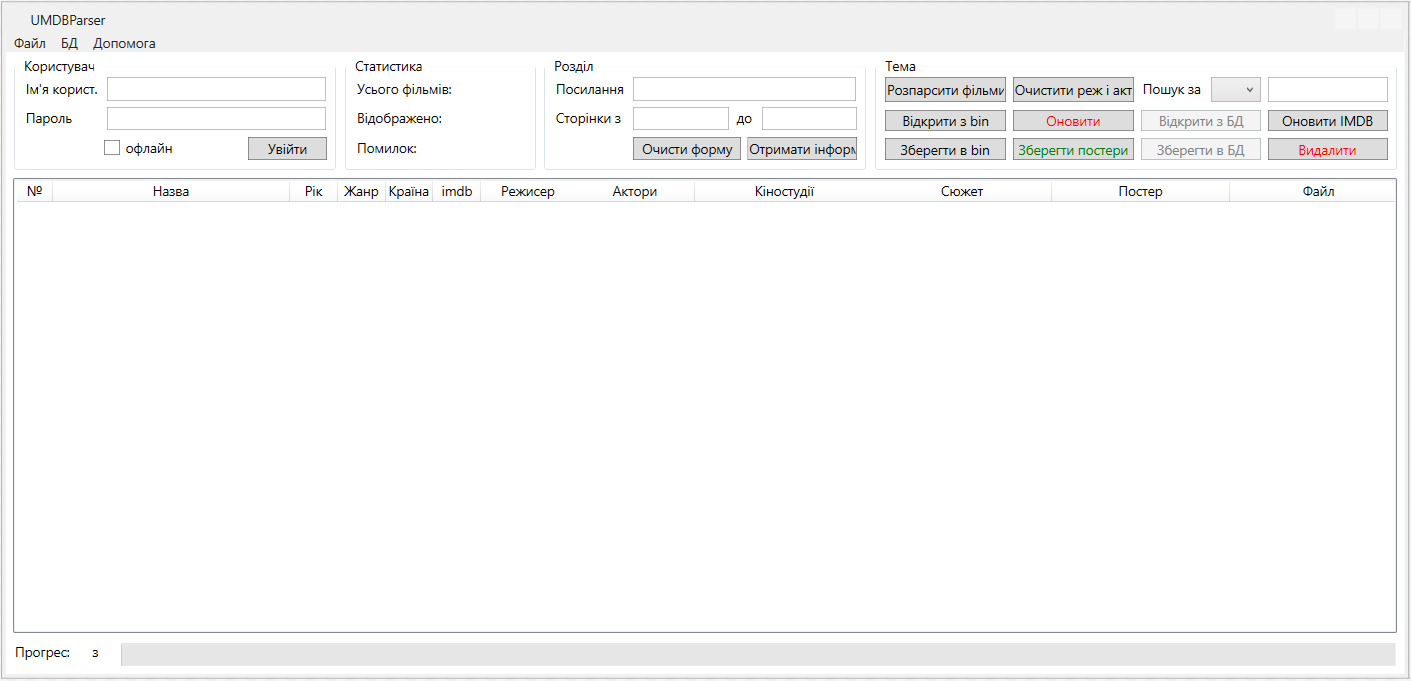
Отож згідно даної структури модель фільму міститиме наступні поля:

* Назва фільму (українською та оригінальна назви)
* Рік виходу фільму в прокат
* Перелік жанрів
* Перелік країн
* Посилання на відповідну сторінку з сайту imdb.com
* Перелік кінокомпаній та кіностудій
* Режисер
* Актори
* Сюжет фільму
* Тривалість фільму
* Посилання на постер
* Назва постеру

2.1 Побудова графічного інтерфейсу парсеру

Парсер буде створено з використанням мови програмування C#, технології WPF та патерну проектування MVVM(Model-View-ViewModel). Це дозволить розділити програму на три функціональні частини: графічний інтерфейс, бізнес логіку та саму модель зберігання даних. У WPF графічний інтерфейс будується з використанням XML розмітки.

На ***Рисунку 2.2*** відображено Graphical User Interface(GUI) програми побудований з використанням XML розмітки. Завдяки використанню таких елементів як Grid та DockPanel увесь дизайн є адаптивним і може автоматично підлаштовуватися під будь які розміри екрану комп’ютера.



***Рисунок 2.2*** Основне вікно програми-парсеру

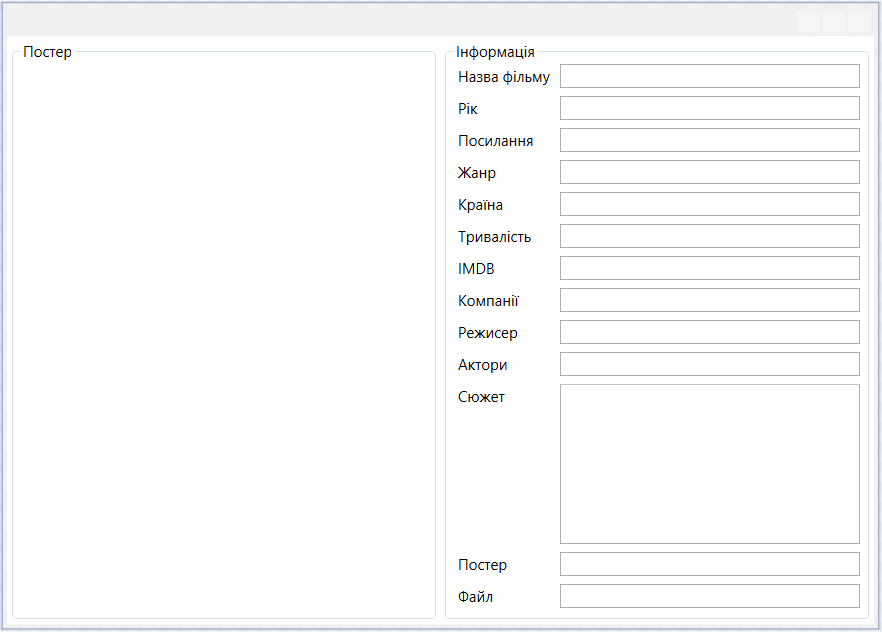
Вікно програми розділено на три частини: меню, функціональна панель та панель відображення списку фільмів.

Меню містить основні функції програми, такі як зберігання та відкривання уже розпізнаних фільмів, збереження та оновлення інформації у базі даних.

Основна панель містить:

1. набір інструментів для логінування користувача(ім’я користувача та його пароль) і ідентифікатор стану користувача(онлайн, офлайн)
2. відображення загальної статистики
3. елементи для вказування посилання на розділ з якого буде братися інформація та діапазон сторінок
4. елементи для парсингу окремо кожного фільму, зберігання всієї інформації у бінарному форматі та її відкривання, оновлення поточного списку новими фільмами, зберігання постерів, функція пошуку та кнопка для видалення обраних фільмів.

Панель для відображення списку фільмів відображатиме інформацію про кожен фільм згідно моделі фільму, щось на кшталт відображення таблички у базі даних. Для надання додаткової функціональності з редагування та перегляду кожного фільму створив також окреме вікно, яке буде відкриватися при клацанні на відповідний фільм у списку.



***Рисунок 2.3*** Вікно для перегляду та редагування інформації про фільм

Для розпізнавання та розділення тексту за певними шаблонами буле використано уже вбудовану у мову C# бібліотеку регулярних виразів Regex.

Щоб мати змогу зручно здійснювати пошук по DOM структурі кожної сторінки без використання складних регулярних виразів було вирішено використовувати сторонню бібліотеку **HtmlAgilityPack**, яка зберігає всю структуру гіпертекствої розмітки сторінки у спеціальні контейнери, завдяки чому може швидко здійснювати пошук за тегами та атрибутами.

2.2 Логінування програми під виглядом звичайного користувача

Як уже згадувалося раніше перегляд значної частини фільмів неможливий без реєстрації, навіть якщо є посилання на фільм то система усе одно переадресує користувача до форми входу.

Аби розв’язати цю проблему потрібно зрозуміти принцип взаємодії між клієнтом і сервером у клієнт-серверній архітектурі. При з’єднанні клієнта із сервером відбувається так-зване рукостискання(handshake) в результаті чого браузер створює спеціальний набір файлів, які називаються Cookie файлами і використовуються сервером для ідентифікації користувача. Якщо користувач увійшов під своїм іменем на певному сайті то сервер надав, а браузер зберіг певний набір ключів та значень, які однозначно ідентифікують користувача. Отож коли користувач здійснить наступний запит на сайт, сервер буде знати, що користувач уже був тут і зможе однозначно ідентифікувати його.

Кожен cookie складається з пари **name=value**, де **name** – це ключ, а **value** – значення ключа. Також кожен cookie файл має певний термін життя, який задається парою **max-age=seconds**, де значення **seconds** – це час життя даного cookie в секундах. Після закінчення терміну життя клієнт має відмовитися від використання cookie, тобто сервер уже не зможе ідентифікувати користувача, а користувач, який використовує застарілі cookie вже не має колишніх повноважень. Зазвичай це значення встановлюється від кількох хвилин до кількох років і протягом цього часу користувач може безперешкодно доступатися до функціональності сайту не вводячи свій логін та пароль повторно. Обмеження в часі були створені аби cookie файли не існували вічно та для захисту користувачів, профілі яких могли бути захоплені зловмисниками. Також цікавим є той факт, що навіть захопивши cookie файли певного користувача і користувач помітив підозрілу активність, він може просто дати запит на знешкодження файлів cookie (вихід з профілю на сайті) внаслідок чого зловмисник теж втратить доступ до профілю.

2.3. Зберігання cookie файлів та здійснення GET запитів

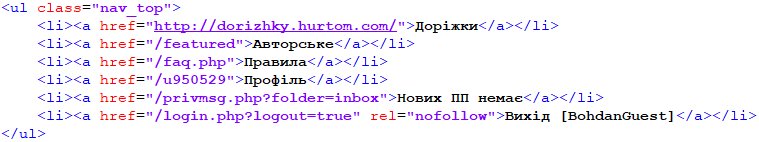
HtmlAgilityPack уже містить функціональність для завантаження html сторінок та зберігання спеціального контейнера CookieCollection, з допомогою якого потім будуть здійснюватися запити. Проте ручне задання кожного ключа і значення є клопітким процесом, а ще через деякий час cookie файли втрачають свою валідність і сервер перестане сприймати запити від програми. Тому було вирішено створити клас для вирішення цієї проблеми і назвати його Session. Цей клас міститиме об’єкт класу HtmlWeb з бібліотеки HtmlAgilityPack, контейнер CookieCollection з файлами Cookie, контейнер із ключами та значеннями полів запиту з форми FormElementCollection(реалізує Dictionary<string,string> та перетворює дані з форми на рядок запиту), методи для відправлення GET та POST запитів з використанням cookie, які повертатимуть об’єкт HtmlDocument. Програма під час надсилання POST запиту зберігатиме контейнер з cookie файлами, а при надсиланні GET запиту тільки використовуватиме.

Для перевірки справності роботи даного класу здійсню POST запит для входу користувача та GET запит для отримання головної сторінки сайту. Про успішність входу свідчитиме кнопка «Вийти» у html коді.

Як можна бачити з ***Рисунків 2.4*** та ***2.5*** це дійсно так і сервер сприймає програму парсер як звичайного користувача, що увійшов в систему.

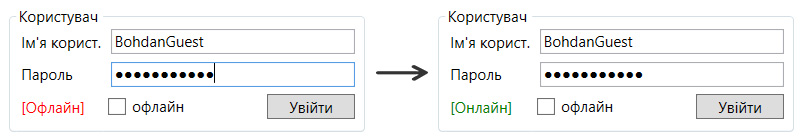


***Рисунок 2.4*** Html код без використання cookie (наявні Зареєстуватися та Вхід)



***Рисунок 2.5*** Html код з використанням cookie (зявилися функції Профіль, Нових ПП немає та Вихід)

Оскільки вхід був успішним і CookieCollection не порожня зміню ідентифікатор [Офлайн] на [Онлайн] у графічному інтерфейсі користувача.



***Рисунок 2.6*** Зміна ідентифікатора під час входу користувача

Тепер можна приступати до парсингу самих фільмів.

2.4 Парсинг фільмів

Оскільки список фільмів зберігається в окремому розділі по 45 фільмів на сторінку, а кожен фільм в окремій темі то розіб’ю парсинг на два етапи:

На першому етапі буду розпізнавати назву фільму, рік виходу фільму та посилання на тему з повним описом фільму.

На другому етапі завантажуватиму окремо сторінку для кожного фільму і діставатиму решту інформації про фільм(жанри, країни, тривалість, посилання на imdb.com, кінокомпанії, режисер, актори, сюжет та посилання на постер).

2.4.1 Перший етап розпізнавання фільмів

Легко можна бачити, що нумерація сторінок лише змінює закінчення посилання додаючи до нього цифри, які використовуються для **LIMIT** у запиті

<https://toloka.to/f70> 1-ша сторінка

<https://toloka.to/f70-45> 2-га сторінка

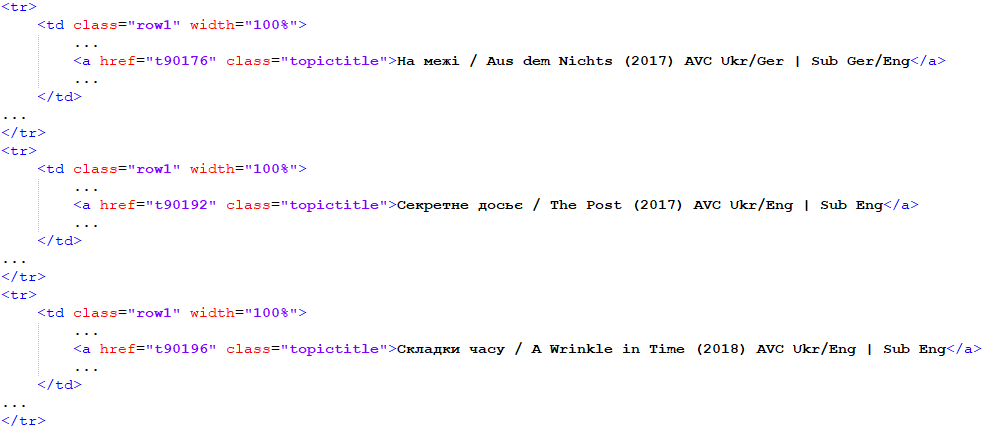
<https://toloka.to/f70-90> 3-тя сторінка

…………

Отож для кожної сторінки в циклі легко порахувати її посилання, просто скориставшись формулою

Таким чином задавши лише посилання на відповідний розділ та проміжок сторінок з якої по яку хочемо отримати інформацію можна переглянути усі ці сторінки у простому циклі з параметром.

Переглянувши html код можна зауважити, що всі посилання на фільм та назва знаходяться в середині елементів **<a>** з атрибутом **class="topictitle"**



***Рисунок 2.7*** Html структура розділу з фільмами

Для завантаження html сторінки буде використано метод GET з класу Session, який завантажить сторінку для нібито зареєстрованого користувача.

А з використанням бібліотеки HtmlAgilityPack і методу SelectNodes з параметром

SelectNodes("//a[@class='topictitle']")

Отримаю колекцію всіх елементів з класом topictitle.

Далі в циклі для кожного елемента просто дістаю значення атрибуту **href** використовуючи метод GetAttributeValue("href") і отримую посилання на повний опис фільму. Щоб отримати назву достатньо викликати властивість **InnerText**, яка витягне увесь тест всередині тегу <a>. Проте ця назва містить не тільки саму назву фільму, а й рік та додаткову інформацію про фільм. Для відокремлення назви та визначення року скористаюся регулярним виразом, адже рік знаходиться в дужках.

Regex.Match(name, @"(\([0-9]{4}\))")

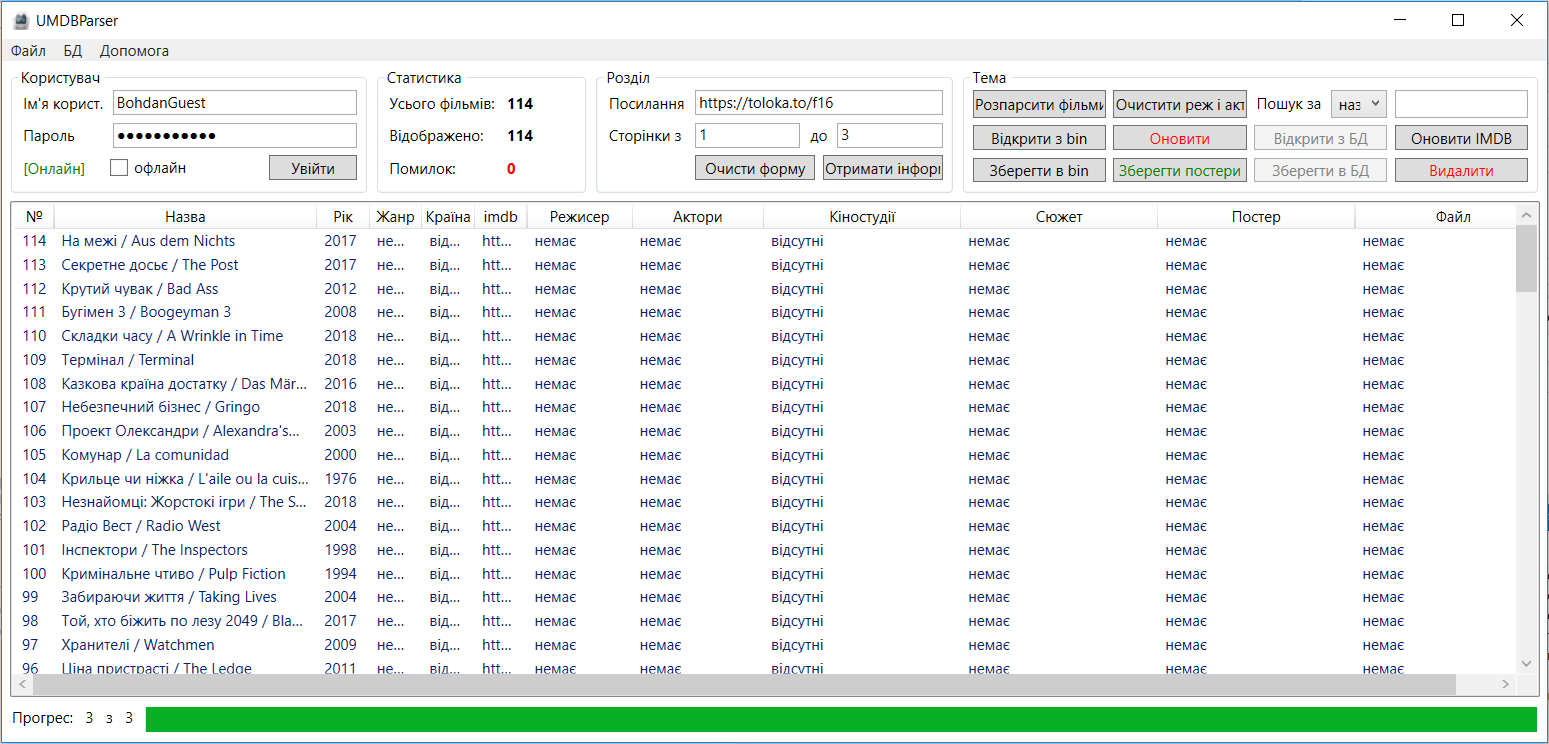
де name – повна назва з роком

Даний регулярний вираз шукає чотири повторення цифр від 0 до 9 записаних у дужках. Весь текст до цього – це назва фільму(українська + оригінальна), а отриманий результат регулярного виразу – рік. Також вважається, що назва фільму унікальна, а повтори у результат не включаються. За виключення повторів відповідає реалізація інтерфейсу IEqueatable<Movie> та методу Equals, який потім використовується для перевірки в методі Contains. Сам метод Contains викликається щоразу при спробі додати новий фільм і якщо такий фільм уже існує та він пропускається, а перевірка застосовується до наступного знайденого фільму.

Варто зауважити, що оскільки завантаження сторінки та її обробка займають деякий час та багато ресурсів то на час отримання інформації неактивним стає графічний інтерфейс користувача і немає змоги нічого переглядати чи виконувати до завершення всього циклу отримання інформації. Тому було вирішено використати багатопотоковість мови програмування C#, аби розподілити отримання сторінки та її розпізнавання на два окремих процеси, що дало змогу вільно переглядати процес виконання та уникнути заблокованості графічного інтерфейсу. В даному випадку метод для отримання інформації GetAllMovies було позначено за допомогою специфікатора async, а частину коду з завантаженням сторінки блоком await. Таким чино графічний інтерфейс не блокується і є змога переглядати інформацію про фільми, які вже було розпізнано.

Для тестування витягну фільми з перших 3-х сторінок. З **Рисунку 2.8** видно, що все працює правильно, адже система витягнула 114 фільмів(очевидно були видалені повтори), назву було відділено від року, а рік збережено окремо як цілочислове значення. Уся інша інформація поки що відсутня.

Саме посилання на фільм збереглося у моделі, але не відображається на графічному інтерфейсі головного вікна, його можна переглянути лише під час перегляду детальнішої інформації.



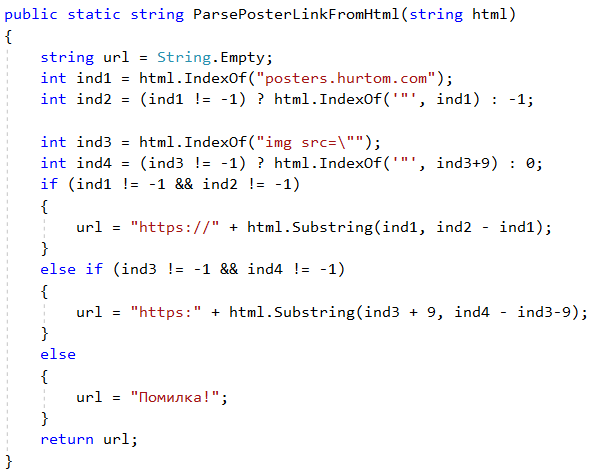
***Рисунок 2.8*** Перший етап парсингу фільмів

2.4.2 Другий етап розпізнавання фільмів

На другому етапі будую конкатенацію посилання <https://toloka.to> та отриманого на першому етапі посилання і завантажую сторінку з детальною інформацією. Метод для виконання другого етапу назву **ParseMovies**. З html коду кожної теми з фільмом можна побачити, що всі повідомлення знаходяться в середині тегів **<span>** з атрибутом **class="postbody".** Використовуючи метод SelectSingleNode з параметром

SelectSingleNode("//span[@class='postbody']") отримаю лише перше повідомлення, яке й містить детальну інформацію про фільм.

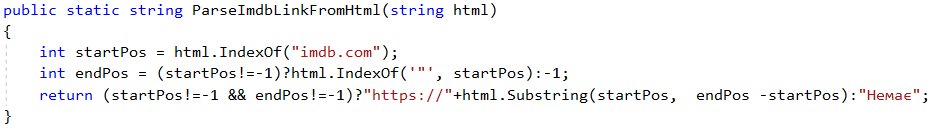
Спершу знайду посилання на постер до фільму, для цього створю статичний метод **ParsePosterLinkFromHtml** у класі **Movie**, який буде шукати перше входження посилання на дочірній сайт posters.hurtom.com, або якщо такого немає то першу картинку, яка зустрічається. Алгоритм працює саме так, тому що не всі постери зберігаються безпосередньо на дочірньому сайті, особливо це стосується старіших фільмів.



***Рисунок 2.9*** Метод ParsePosterLinkFromHtml

Також створю статичний метод **CreatePosterFileName**, який на основі назви фільму, року та посилання формуватиме назву файлу для кожного постеру. Оскільки назва файлу не допускає символи такі як ':','\*','?', '"', '<', '>' то заміню їх в назві на пробіл, а символ '/' на '-' і додавши до назви рік та розширення файлу отримаю назву постеру.

Далі також необхідно знайти посилання на сайт **imdb.com**, для цього створю ще один статичний метод **ParseImdbLinkFromHtml** у класі Movie з наступним кодом.



***Рисунок 2.10*** Метод ParseImdbLinkFromHtml

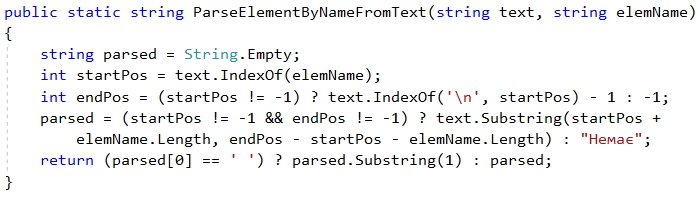
Алгоритм шукає перше входження рядка imdb.com, якщо таке існує і повертає повне посилання, або інформацію про те, що його немає.

Оскільки значна частина параметрів фільму знаходиться у тому ж рядочку, що й сам параметр то для пошуку за такими ключовими назвами як жанр, країна та інші також знадобиться статичний метод **ParseElementByNameFromHtml** для пошуку за ключовим словом. Цей метод шукає перше входження елементу(наприклад «Жанр:») та повертає як результат увесь рядок до кінця, адже з наступного рядка починається інший атрибут.

Особливістю роботи цього методу є те , що для його роботи необхідно видалити всю html розмітку використавши регулярний вираз.

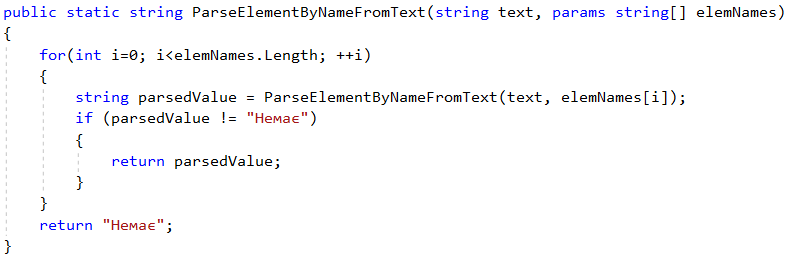
Regex.Replace(input, "<.\*?>", "");

Цей вираз замінить усі html теги на порожній рядок, тому всі розпізнавання такого виду виконуються аж після пошуку сюжету, про який буде сказано пізніше.



***Рисунок 2.11*** Метод ParseElementByNameFromText

Оскільки можуть бути випадки, коли в одному фільмі пише Країна, в іншому - Країни, в одному - Кіностудія, в другому - Кінокомпанія і т.д. тому перевантажу цей метод, аби була змога передавати велику кількість параметрів для пошуку.



***Рисунок 2.12*** Перевантаження методу ParseElementByNameFromText

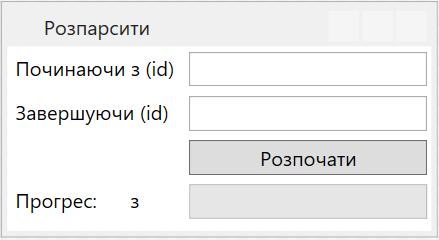
Для пошуку сюжету всього вище перерахованого буде недостатньо, адже опис сюжету починається з наступного рядочку після слова «Сюжет», а попередні алгоритми розраховані лише на випадки, коли інформація знаходиться у тому ж рядочку**.** Щоб отримати сюжет фільму доведеться врахувати той факт, що сюжет розтягується на весь рядочок з використанням стилю text-align:justify; та є курсивним. Скористаюся наступним регулярним виразом.

Regex.Match(post, "<div style=\"text-align: justify;\"><span style=\"font-style: italic\">(.|\n)\*?</span></div>"), де post – html код першого повідомлення з теми

Цей вираз знайде текст, який буде знаходитися в середині такої html конструкції включно з переходами на новий рядок.

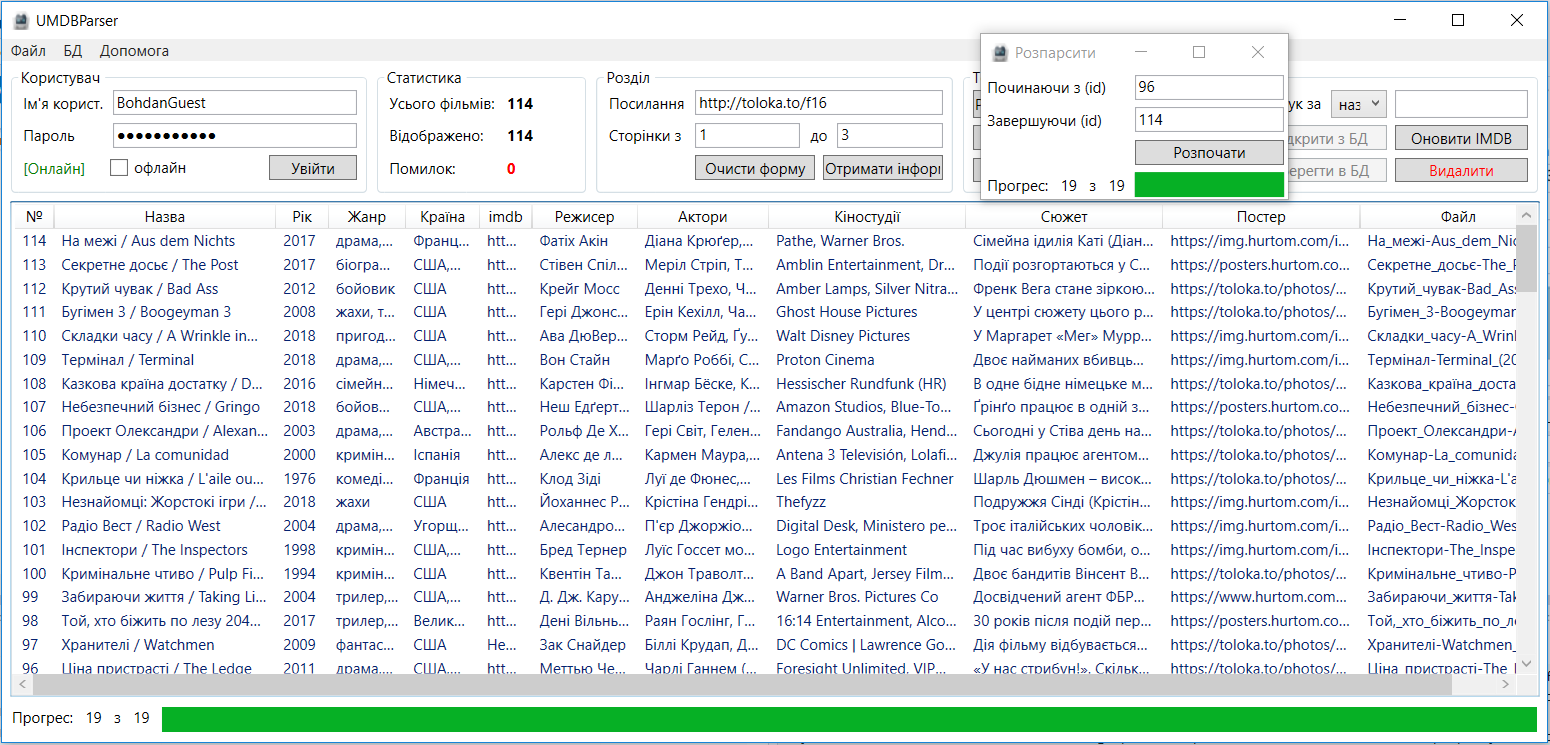
Як і у випадку із методом **GetAllMovies** частина коду у поточному методі **ParseMovies** також було позначено за допомогою await блоку, а сам метод специфікатором async. Це дало змогу розблокувати інтерфейс користувача на час отримання інформації та можливість переглядати уже отриману інформацію.

Також для зручності вказування проміжку з якого по який фільм хочу отримати інформацію створю додаткове вікно під назвою **ParsePagesWindow**



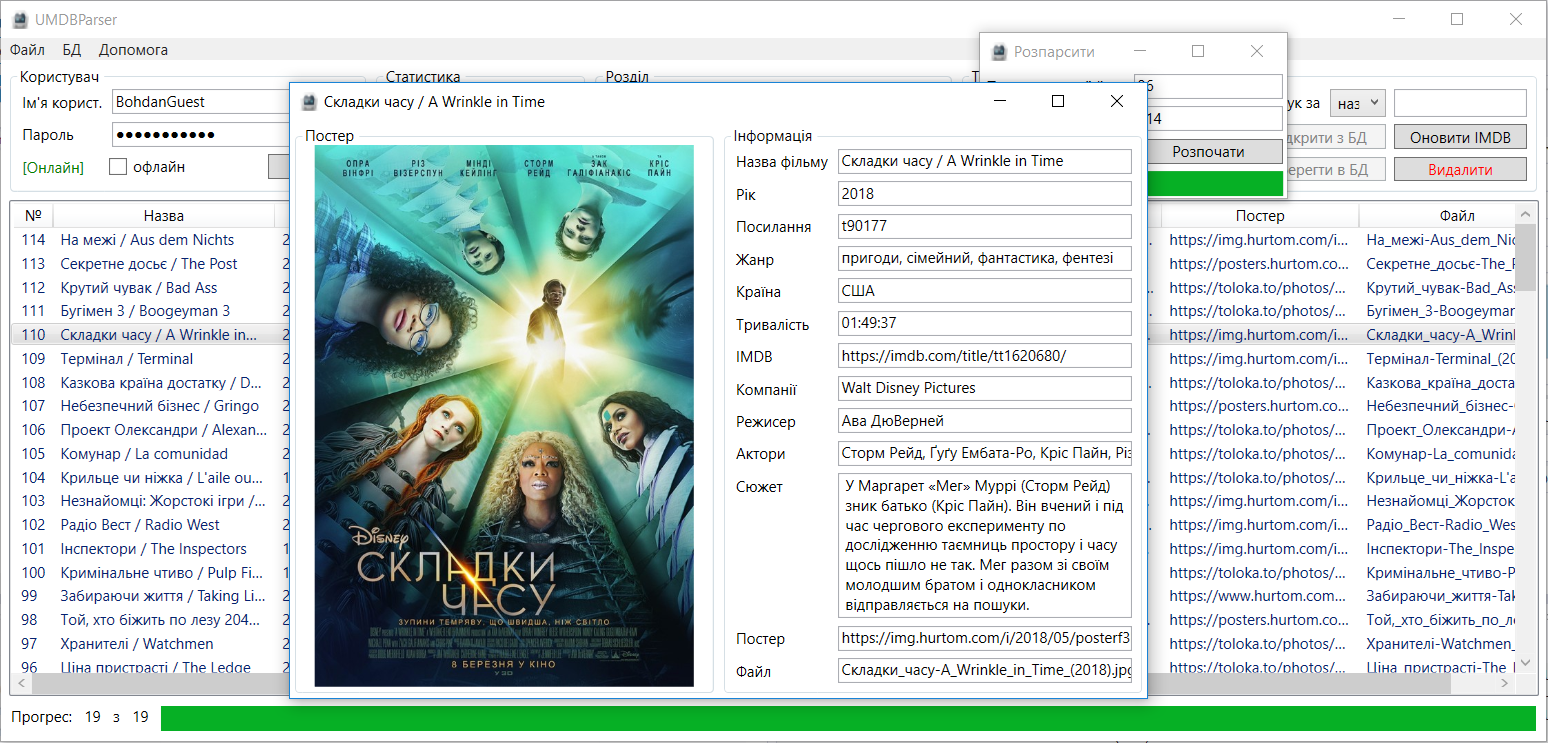
***Рисунок 2.13*** Вікно для парсингу фільмів з певного проміжку

Протестую другий етап отримання інформації розпізнавши фільми з 96 по 114, результат продемонстровано на ***Рисунку 2.14.***



***Рисунок 2.14*** Другий етап парсингу фільмів

Як бачимо парсинг працює чудово і вся інформація отримана правильно. Також можна переглянути та редагувати інформацію про кожен фільм просто клацнувши по фільмі, внаслідок чого поточна модель передасться у **MovieWindow** і завдяки атрибуту **UpdateSourseTrigger=PropertyChanged** оновлюватиметься при зміні будь якого поля у вікні редагування. Завдяки бібліотечному класу **BitmapImage** постер до фільму завантажується з посилання на постер автоматично та відображається на графічному інтерфейсі користувача.



***Рисунок 2.15*** Перегляд детальної інформації про фільм

2.5 Розпізнавання жанру та країни виробництва для окремого фільму

Оскільки перелік жанрів та країн є скінченним та знаходиться кожен в окремому об’єкті типу string було б добре створити два словники для жанрів та країн, які в майбутньому будуть використані для фільтрації пошуку на сайті. Легко побачити, що перелік жанрів та країн розділяється символом ',' та пробілами, тому для розпізнавання окремо жанрів та країн створю дві додаткові моделі Country(Id, Name) – містить унікальний ідентифікатор і назву країни та Genre(Id, Name) – також містить унікальний ідентифікатор для кожного жанру і його назву. Для цих обох класів буде реалізовано інтерфейс IEquatable, щоб виключити повторення жанрів та країн в словниках. Також зміню модель фільму додавши до неї дві властивості List<Genre> та List<Country>.

Щоб згенерувати жанри та країни з усіх фільмів необхідно в циклі перебрати усі фільми, а далі дотриматися наступного алгоритму:

Спершу рядок жанру та список країн розбити на масив стрінгів використовуючи бібліотечний метод Split(). Далі для кожного елемента в масиві застосувати два наступні регулярні вирази:

Regex.Replace(input, @"^\s+", "");

Regex.Replace(input, @"\s+$", "");

Перший вираз видалить усі пробіли до початку слова, якщо такі існували, а другий усі пробіли з кінця. Також для жанру необхідно застосувати ще один регулярний вираз, оскільки деякі жанри на сайті можуть починатися з великої літери.

Regex.Replace(genre, @"[А-Я]", с => с.ToString().ToLower());

Даний вираз замінить усі великі входження літер від А до Я на їх малі відповідники, аби наш словник не збільшувався лише за рахунок того, що деякі жанри написані з великої літери. Для країн немає потреби застосовувати цей же принцип, оскільки всі назви країн уже задані з великої літери і не будуть створювати зайві елементи у словнику.

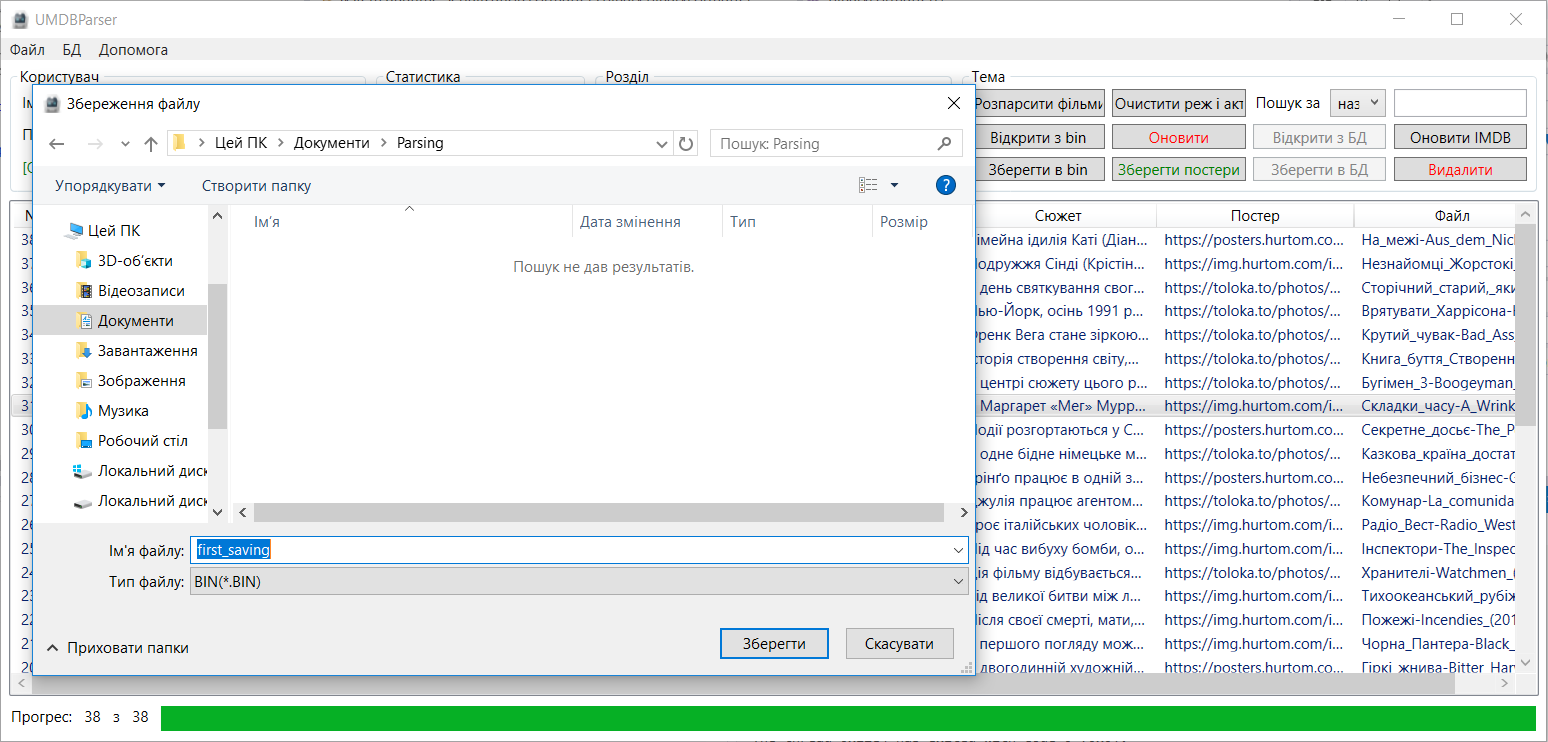
Таким чином перебираючи усі фільми одночасно генеруються словники жанрів та країн і додаються відповідні жанри та країни до кожного фільму окремо.

2.6 Зберігання інформації в бінарному форматі

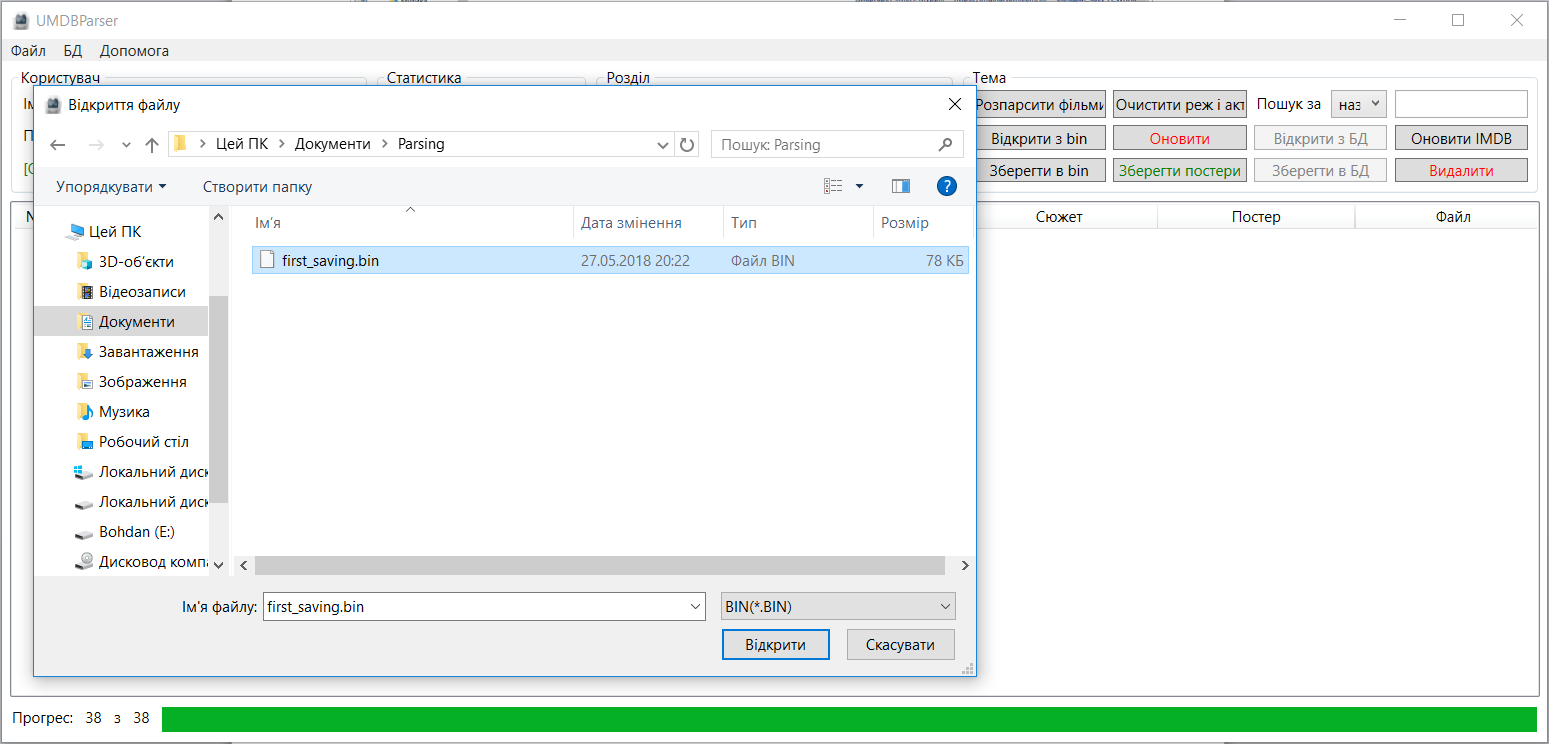
Оскільки остаточне розпізнавання всіх фільмів буде здійснюватися частинами, а не все одразу, аби не перевантажувати сервер з якого беремо інформацію, було вирішено тимчасово зберігати дані у бінарному форматі. Бінарний формат обрано тому, що він найзручніший для зберігання великого обсягу інформації, для порівняння зберігання в XML форматі збільшило б розмір кінцевого файлу приблизно на 25%.

Для зберігання даних у бінарному форматі додам до моделі фільму конструктор за замовчування та атрибут [Serializable]. Слід зауважити, що поля з жанрами і країнами для кожного фільму також повинні бути збережені у бінарному форматі, тому до моделі жанру та країни також додам відповідний атрибут і конструктор за замовчуванням. За саме зберігання у бінарний формат буде відповідати об’єкт BinaryFormatter, який уже вбудовано у бібліотеку для сереалізації фреймворку .NET Framework.

Скориставшись стандартними діалоговими вікнами для зберігання та відкривання файлів проведу перевірку на правильність зберігання у бінарному форматі.



***Рисунок 2.16*** Зберігання у бінарному форматі



***Рисунок 2.17*** Відкриття та десереалізація бінарного файлу



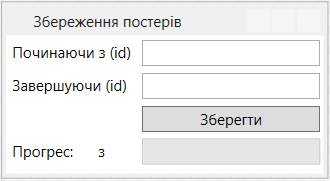
***Рисунок 2.18*** Результат десереалізації бінарного файлу

Отож як видно з ***Рисунку 2.18*** в результаті збереження та відкриття інформації з бінарного файлу усі дані збереглися, а загальний розмір файлу з розширенням bin складає всього лиш 78 КБ для 38 фільмів.

2.7 Завантаження постерів

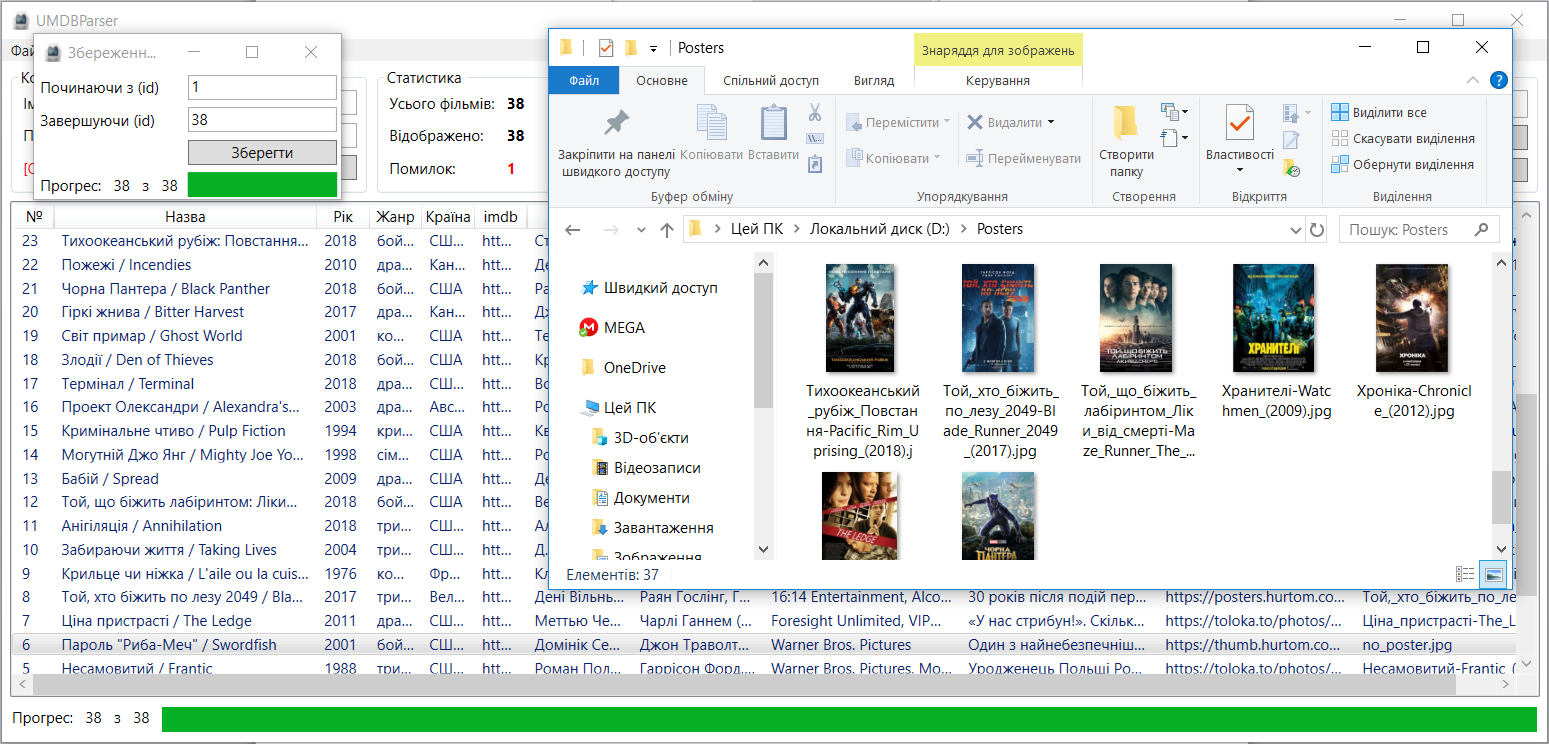
У попередніх пунктах для перегляду окремого постера до кожного фільму потрібно було відкривати детальну інформацію про кожен фільм і програма сама завантажувала постер з посилання і відображала його. Для відображення інформації разом з постерами на веб-сайті необхідно завантажити усі постери у окремий каталог та зберегти посилання на кожний постер у базі даних, аби щоразу не було потреби звертатися до сторонніх ресурсів за постером.

Як уже згадувалося у пункті **2.4** було створено статичний метод **CreatePosterFileName,** який для кожного фільму за його назвою, роком та посилання на постер генерує назву файлу. Все, що залишається це зберегти картинку з посилання у файл з відповідно згенерованою назвою. Для завантаження скористаюся бібліотечним класом **WebClient**, який містить метод **DownloadFile**(string address, string fileName). Цей метод дозволяє завантажити файл за його посиланнями у файл з назвою fileName. Щоб вибрати каталог куди зберігати усі постери використаю стандартне діалогове вікно для вибору папки. Також для зручності завантаження постерів з певного проміжку фільмів було створено вікно **SavePostersWindow**, яке дуже схоже на вікно **ParsingPagesWindow** і має схожу функціональність, проте замість отримання інформації про фільми зберігає постери.



***Рисунок 2.19*** Вікно для збереження постерів з певного проміжку

В методі було передбачено випадок, коли постер на сайті відсутній, адже для фільмів, які були додані раніше постери могли не зберегтися, або ресурси на яких вони зберігалися припинили свою роботу. Тому для уникнення помилок під час завантаження у випадку, коли файл не доступний буде виникати виняткова ситуація, обробник для якої автоматично замінить назву файлу на **no\_poster.jpg** (стандартна картинка з написом про те, що постер відсутній). Проведу тестування завантаживши усі 38 постерів до фільмів.



***Рисунок 2.20*** Результат збереження постерів

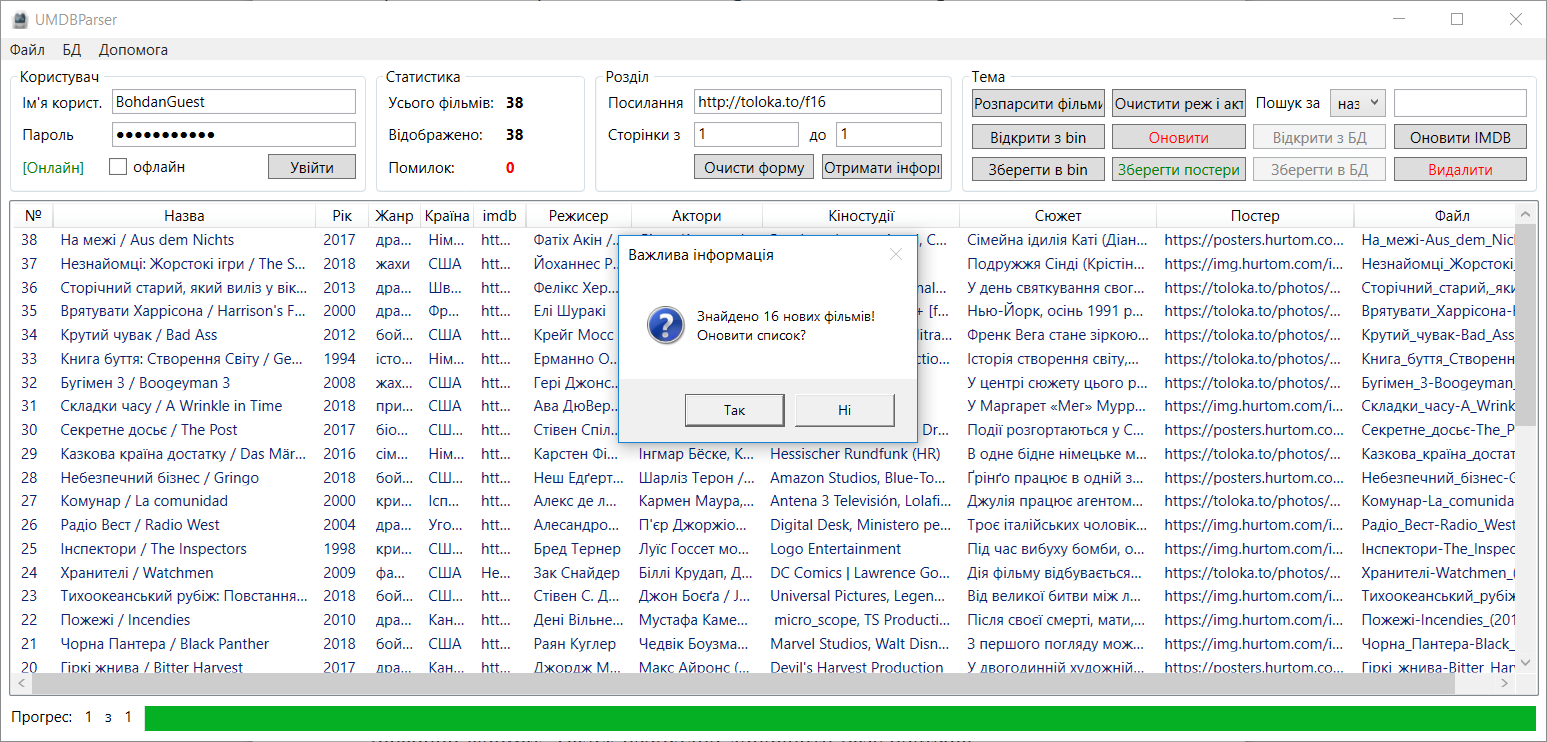
На ***Рисунку 2.20*** з 38 постерів успішно збережені 37 і виникла одна помилка під час збереження постеру до фільму 2001 року, але система успішно замінила назву файлу на **no\_poster.jpg**.

2.8 Оновлення бази фільмів

Оскільки майже щодня на сайті з’являються нові фільми, потрібно також створити функціональність, яка б давала змогу оновлювати поточну базу фільмів. Для доповнення метод **Update** буде виконувати схожу функціональність, як і метод для початкового отримання інформації на першому етапі парсингу фільмів(**GetAllMovies**), а далі кожен фільм буде розпізнаватися окремо уже з використанням готового другого етапу(**ParseMovies**). Єдиною відмінністю між першим етапом отримання інформації та її оновленням буде перевірка на те, чи часом уже не існує такий фільм у поточній базі. Також система буде інформувати користувача про кількість фільмів, які було додано. Оскільки можливий випадок, коли станеться збій в роботі програми нові фільми будуть зберігатися в окремій колекції і лише після підтвердження від користувача додаватися до основної бази фільмів.

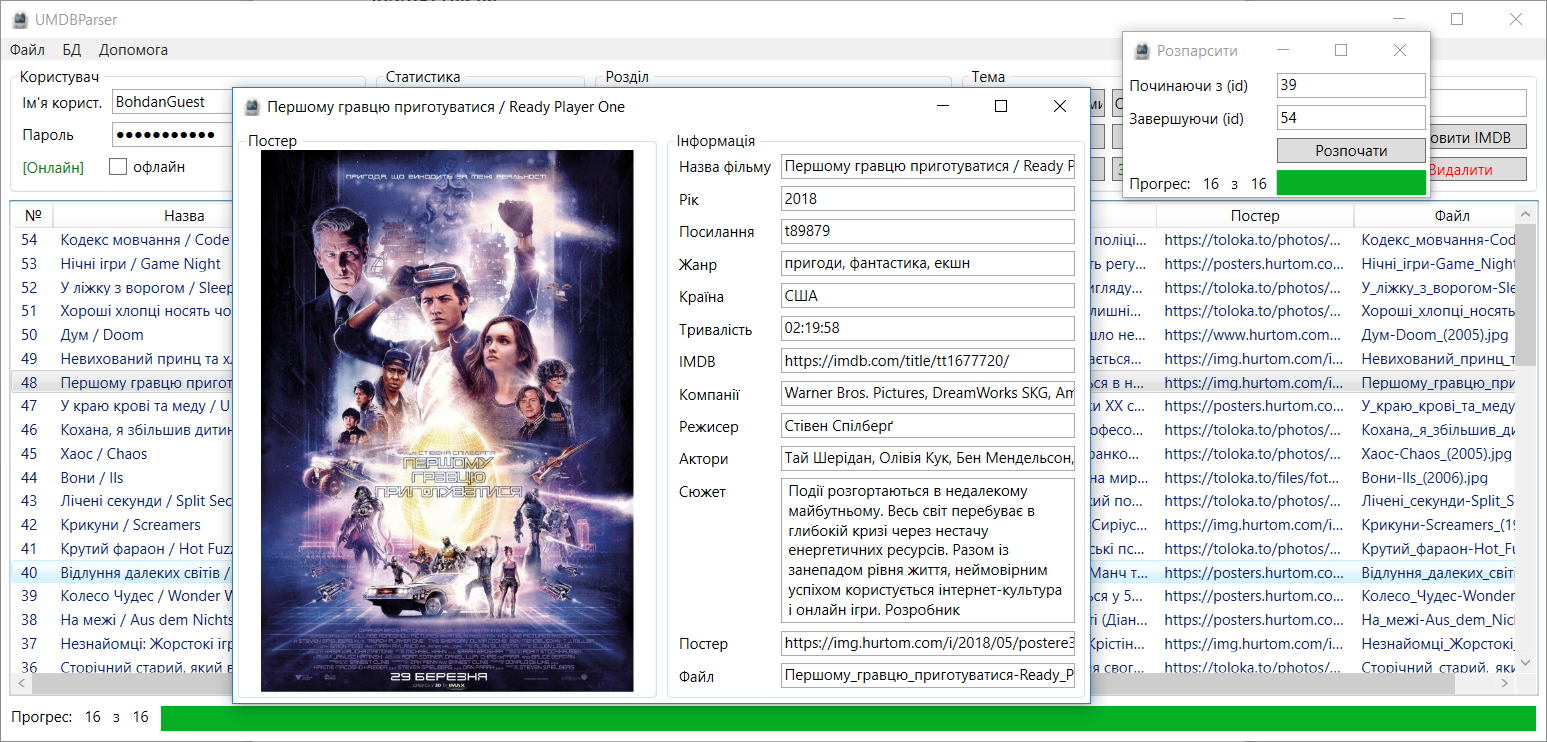
Як і метод **GetAllMovies** поточний метод також буде виконуватися з використанням багатопотоковості, щоб не блокувати графічний інтерфейс користувача.

Протестую метод оновлення, доповнивши тестову базу в якій було 38 фільмів.



***Рисунок 2.21*** Доповнення бази даних

Як бачимо, методу вдалося знайти 16 нових фільмів, які були додані на сайт впродовж кількох попередніх днів. Після того, як я дав програмі дозвіл на додавання нових фільмів до поточної бази виконаю другий етап розпізнавання і отримаю детальну інформацію про кожен фільм та збережу оновлену інформацію у бінарний формат. Також доповню базу постерів новими.



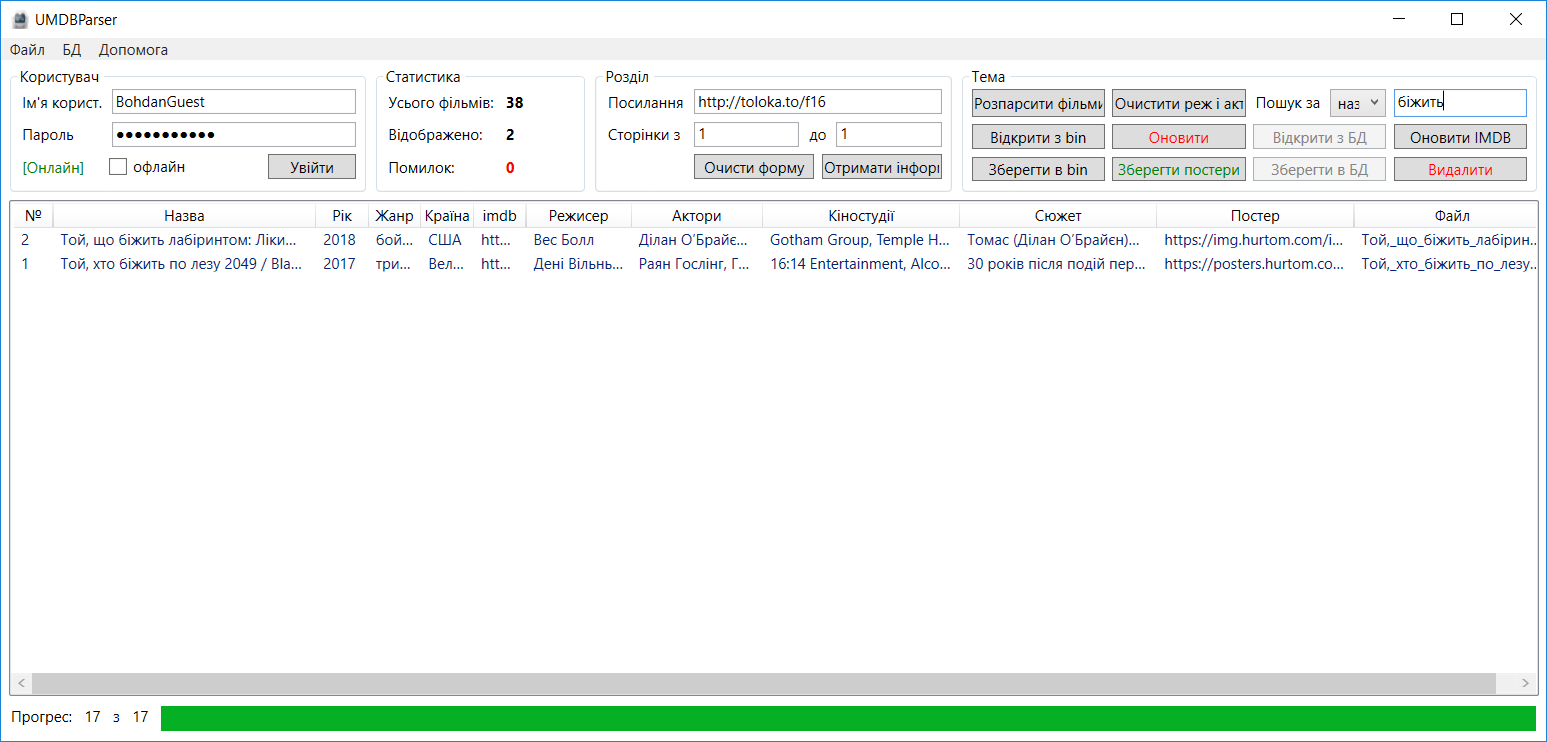
***Рисунок 2.22*** Отримання всієї інформації про фільми

Отож як показано на ***Рисунку 2.22*** метод успішно доповнив базу 16-ма новими фільмами і в результаті отримали 54 фільми всього.

2.9 Пошук за різними критеріями

Для зручності роботи та навігації програмою під час великого обсягу даних було вирішено створити простий пошук за назвою фільму, роком, режисером та іншими властивостями моделі фільму.

Структура патерну MVVM дає змогу зручно реалізувати пошук, який буде оновлювати в режимі реального часу при зміні критерію пошуку. Це все завдяки атрибуту **UpdateSourceTrigger=PropertyChanged**, який буде викликати пошук, як тільки зміниться вміст поля пошуку. Для ефективного пошуку використаю метод Contains() з бібліотечного класу String, який дасть змогу шукати фільми за критерієм того, чи містить певне поле ту чи іншу назву(частина назви фільму, рік і тд).



***Рисунок 2.23*** Пошук за ключовим словом «біжить»

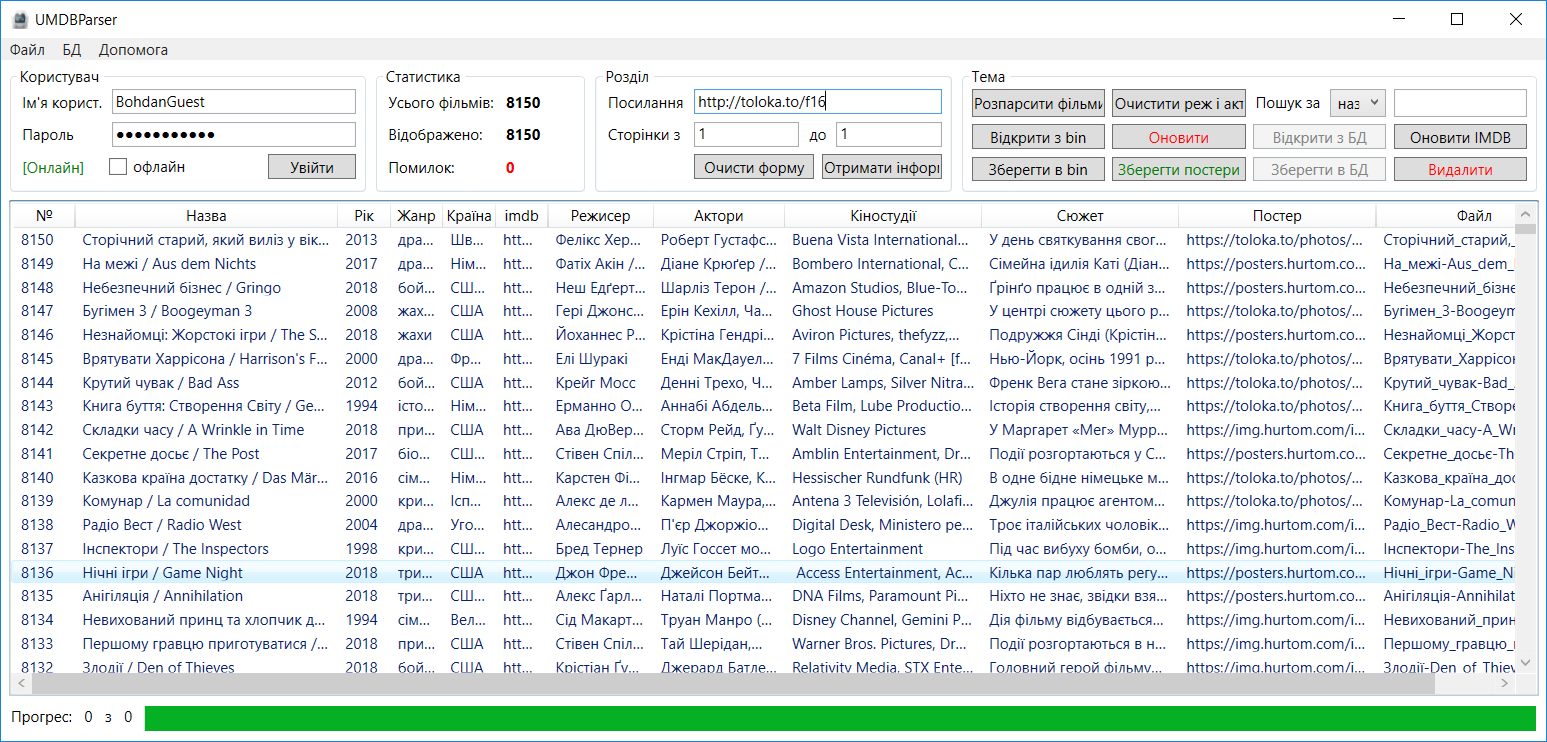
2.10 Розпізнавання та зберігання всіх фільмів та постерів.

У загальному сайт hurtom.com містить 234 сторінки в розділі фільми. За приблизними підрахунками це 10530 фільмів, але значна частина з них повторюється, тому остаточна цифра буде меншою.

Спочатку отримаю інформацію про фільми зі всіх 234 сторінок, скориставшись першим етапом розпізнавання. В результаті отримав 8150 фільмів. Через велику кількість фільмів парсити всі одночасно небезпечно, оскільки це може перевантажити сервер і призвести до його несправності. Тому було вирішено парсити по 2000 фільмів кожного дня. Розпізнавання одного фільму триває від 800 мс. до 1 секунди 200 мс. При цьому парсеру потрібно щодня від 40 до 45 хвилин на отримання всієї інформації і 4 дні на розпізнавання всієї інформації. В кінцевому результаті парсер витратив ~3 годин часу на розпізнавання всіх 8150 фільмів, як це показано на ***Рисунку 2.24***. Це набагато швидше ніж людина зробила б вручну. Для даного списку фільмів також було згенеровано список жанрів, розмір якого склав 163 жанри та список країн, яких виявилося 135. Після збереження у бінарний файл, його розмір склав 14,7 МБ, що досить непогано для такої великої кількості інформації.

Далі необхідно завантажити усі постери, які були розпізнані для кожного фільму. Загалом було завантажено 7713 постерів, а 437 постерів були нерозпізнані чи не знайдені та замінені на напис про відсутність постеру. Розмір всіх постерів складає 3 ГБ та 20 МБ, а час витрачений на їх завантаження 3087 секунд = 51 хвилині з середньою швидкістю завантаження 10 Мб/c = 1 МБ/c.

Отож загальний час на розпізнавання та збереження всіх фільмів та постерів склав 3 години 51 хвилину.



***Рисунок 2.24*** Результат загального парсингу усіх фільмів

# Побудова web-орієнтованої системи

3.1 Модель бази даних

Для бази даних буде створено три основні таблички – це фільми(Movies), жанри (Genres) та країни (Countries):

* Таблиця **Movies**(Id, Name, Year, ImdbLink, Director, Actors, Companies, Story, Poster, Length) містить унікальний ідентифікатор кожного фільму, його назву, рік виходу в прокат, посилання на сайт imdb.com, режисера, кінокомпанії, сюжет фільму, його постер та тривалість фільму.
* Таблиця **Genres**(Id, Name) містить унікальний ідентифікатор для кожного жанру та його назву.
* Таблиця **Countries**(Id, Name) містить унікальний ідентифікатор для кожної країни та назву країни.

Щоб зв’язати табличку фільмів з табличками Genres та Countries буде використано також дві проміжні таблички:

* Таблиця **MovieGenres**(MovieId, GenreId) – зв’язує ідентифікатор фільму з ідентифікатором жанру
* Таблиця **MovieCountries**(MovieId, CountryId) – зв’язує ідентифікатор фільму

Також будуть додані таблички для зберігання користувачів та їхніх ролей:

* Таблиця **Users**(Id, Email, Name, UserName, PasswordHash)

Містить унікальний ідентифікатор для кожного користувача його email-адресу, ім’я, нікнейм на сайті та хеш паролю.

* Таблиця **Roles**(Id, Name)

Містить унікальний ідентифікатор для кожної ролі та назву самої ролі(рангу) користувача.

* Таблиця **UserRoles**(UserId, RoleId)

Містить ідентифікатор користувача, який має певну роль та ідентифікатор самої ролі користувача.

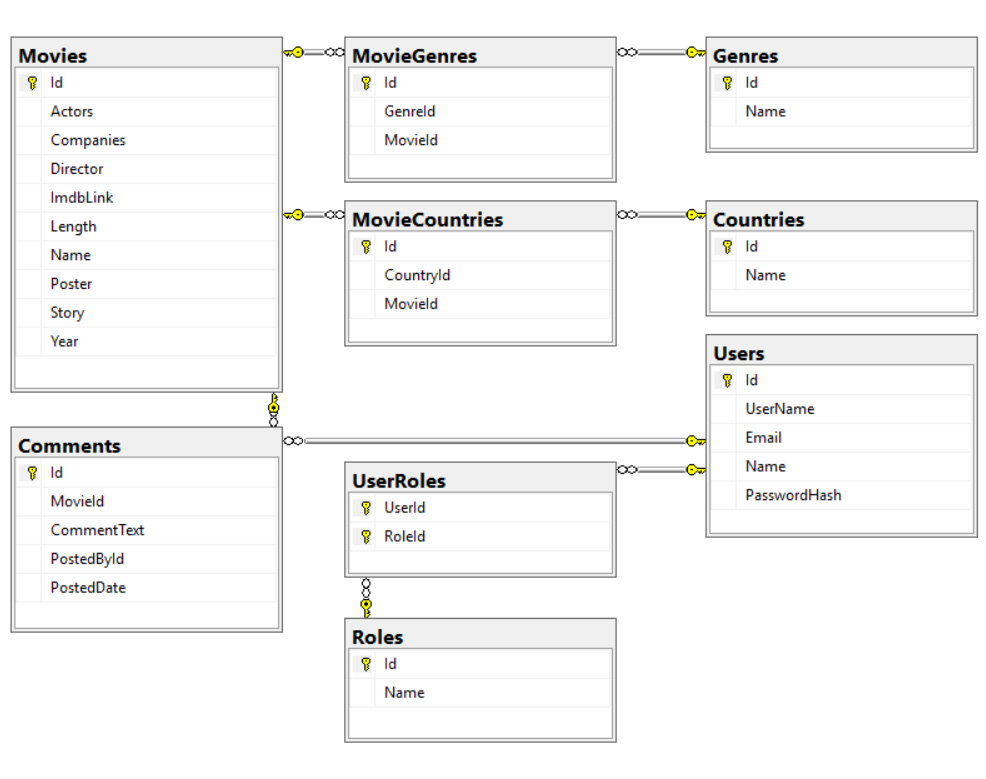
Активність користувачів на сайті буде визначатися ще однією табличкою:

* Таблиця **Comments**(Id, MovieId, CommentText, PostedById)

Містить унікальний ідентифікатор кожного відгуку, ідентифікатор фільму у якому був залишений відгук, текст коментаря та ідентифікатор користувача, який його залишив.

3.2 ER-діаграма бази даних

На ***Рисунку 3.1*** зображена ER-діаграми бази даних зі всіма табличками та зв’язками між ними.



***Рисунок 3.1*** ER-діаграма бази даних

Для спрощення роботи з базою даних використаю модуль Entity Framework Core, який містить об’єктно-орієнтований підхід для доступу до бази даних. Використання цього модуля допоможе мапити усі дані з бази даних на колекції C# та пришвидшуватиме роботу пошуку на сайті. Також цей підхід допоможе мені використати уже раніше створені моделі для фільмів, жанрів та країн. Усе, що залишиться – це додати моделі для таблиць MovieGenres, MovieCountries, Users, UserRoles, Roles, Comments та Ratings. Для контролю змін у моделях також використаю міграції, аби мати змогу модифікувати та доповнювати базу даних у разі потреби.

3.3 Заповнення бази даних фільмами

Оскільки базу даних уже створено, то необхідно перенести до неї всі фільми, які були розпізнані парсером. Щоб не встановлювати модуль Entity Framework Core для програми-парсеру використаю бібліотеку ADO.NET тому, що немає потреби мапити усі дані з бази даних, а потрібно всього лиш виконати запити для додавання інформації до бази даних.

Спочатку відкрию всю інформацію з бінарного формату та додам список жанрів та країн до бази даних, для цього скористаюся наступними SQL запитами:

"INSERT INTO Genres(Name) VALUES(N'{0}');", де {0} назва жанру

"INSERT INTO Countries(Name) VALUES(N'{0}');", де {0} назва країни

Оскільки поле Id для країн та жанрів є типу Identity, то воно автоматично буде збільшуватися на одиничку при додаванні нового рядочка до бази даних. Таким чином Id, які містилися у колекціях співпадуть з Id які будуть збережені у базу даних.

Далі додам усі фільми в базу даних скориставшись запитом:

"INSERT INTO Movies(Name, Year, Length, ImdbLink, Companies, Director, Actors, Story, Poster) VALUES(N'{0}',{1},N'{2}',N'{3}',N'{4}',N'{5}',N'{6}',N'{7}',N'{8}')",

де {0}-{9} відповідні властивості з моделі фільму

Також для кожного фільму необхідно додати окремо жанри та країни у проміжні таблички MovieGenres та MovieCountries, для цього використаю ще два запити:

"INSERT INTO MovieGenres(MovieId, GenreId) VALUES ({0},{1});",

де {0} ідентифікатор фільму, який доданий, а {1} – ідентифікатор жанру до фільму

"INSERT INTO MovieCountries(CountryId, MovieId) VALUES({0},{1});"

де {0} ідентифікатор фільму, який доданий, а {1} – ідентифікатор країни виробника фільму

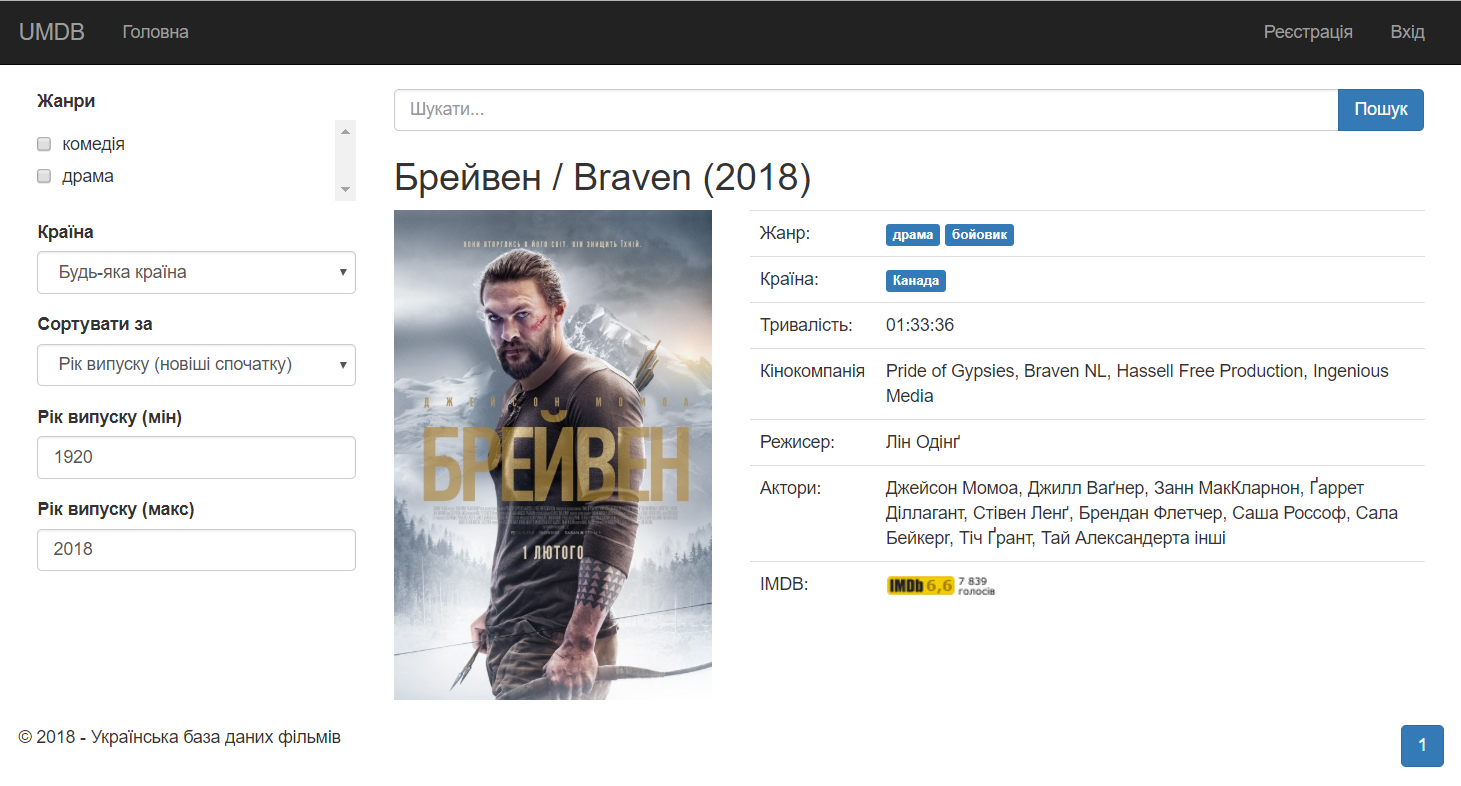
3.4 Графічний інтерфейс

Для побудови графічного інтерфейсу скористаюся вже готовою бібліотекою BootStrap 3.

Кожна сторінка сайту міститиме панель навігації, для переходу на головну сторінку, здійснення пошуку, реєстрації, входу користувача та перегляду свого профілю.

Головна сторінка сайту буде також і сторінкою пошуку. Складатиметься вона з двох панелей: панелі пошуку та панелі результату. Панель пошуку міститиме всі критерії за якими можна здійснювати пошук: жанри, країни, рік випуску фільму(від і до), спосіб сортування та поле для пошуку за назвою. Панель результату буде відображати список усіх фільмів доданих на сайт та коротку інформацію про них. Також внизу панелі буде знаходитися поділ на сторінки. На ***Рисунку 3.2*** продемонстровано реалізацію цього дизайну.

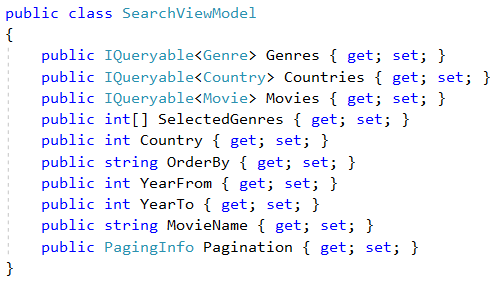
Кожен фільм міститиме власну сторінку, на якій буде можливість залишати коментарі та рейтинг до фільму.



***Рисунок 3.2*** Дизайн сторінки пошуку

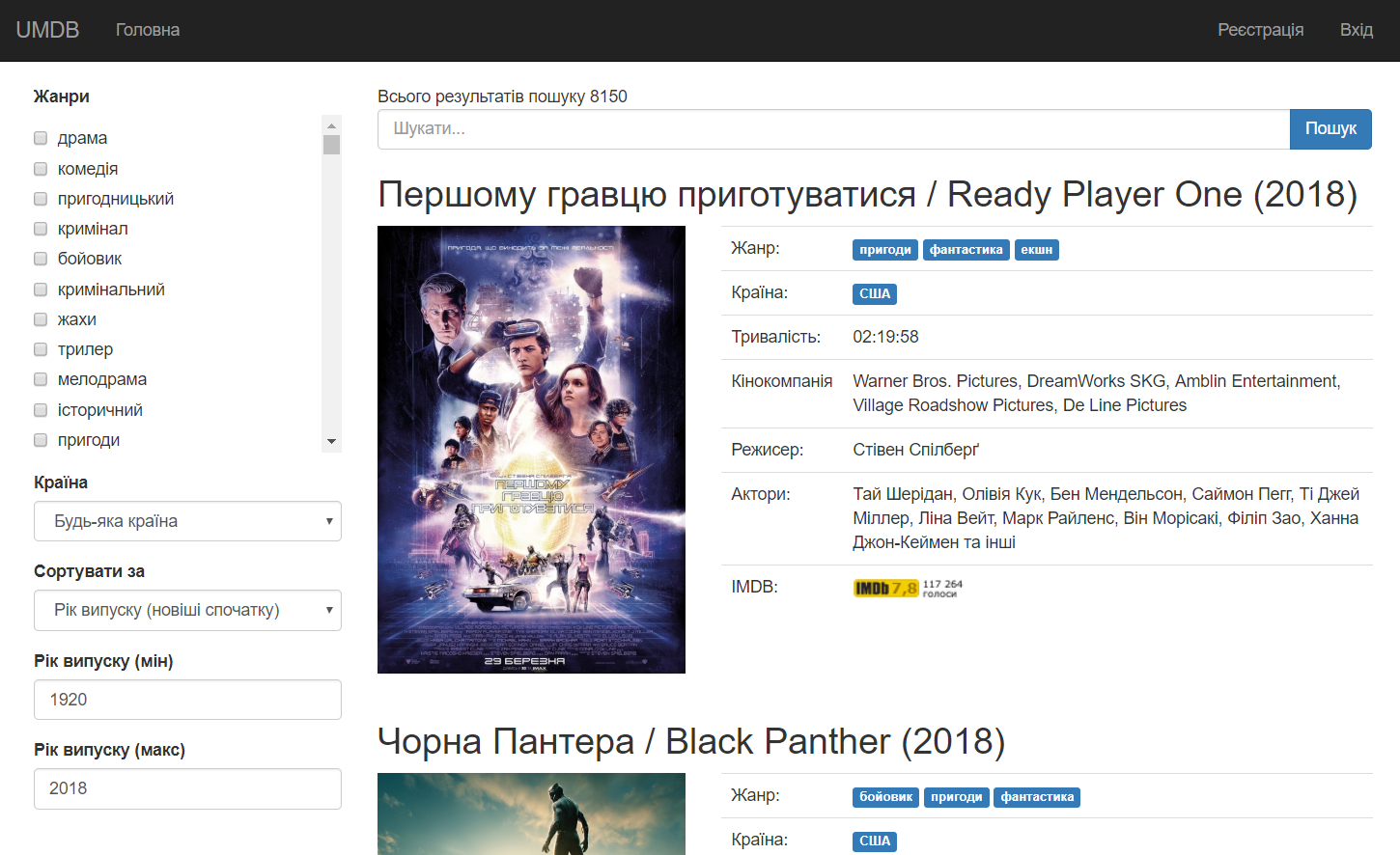
3.5 Виведення даних

Оскільки для побудови web-орієнтованої системи використовується технологія ASP .NET Core MVC то для виведення списку, жанрів, країн та результату пошуку простої моделі даних на кшталт IQueryable<Movie> буде недостатньо. Саме тому я створю додаткову модель під назвою SearchViewModel, яка міститиме окрім полів IQueryable<Movie>, IQueryable<Genre> та IQueryable<Country> також і поля для задання властивостей пошуку, а ще інформацію про сторінку, на якій перебуває користувач. На ***Рисунку 3.3*** зображено структуру цієї моделі.



***Рисунок 3.3*** Структура моделі SearchViewModel

Така структура моделі дасть змогу не лише зберігати у одному класі список жанрів, країн та результати пошуку, а також інформацію про фільтри застосовані до джерела даних і додаткову інформацію про кількість сторінок та поточну сторінку. Всі постери до фільмів перенесу у папку wwwroot/posters, яка буде кореневою для зберігання усіх постерів. Проведу перевірку моделі та графічного дизайну вивівши усі фільми, які були додані до бази даних парсером.



***Рисунок 3.4*** Відображення фільмів на сайті

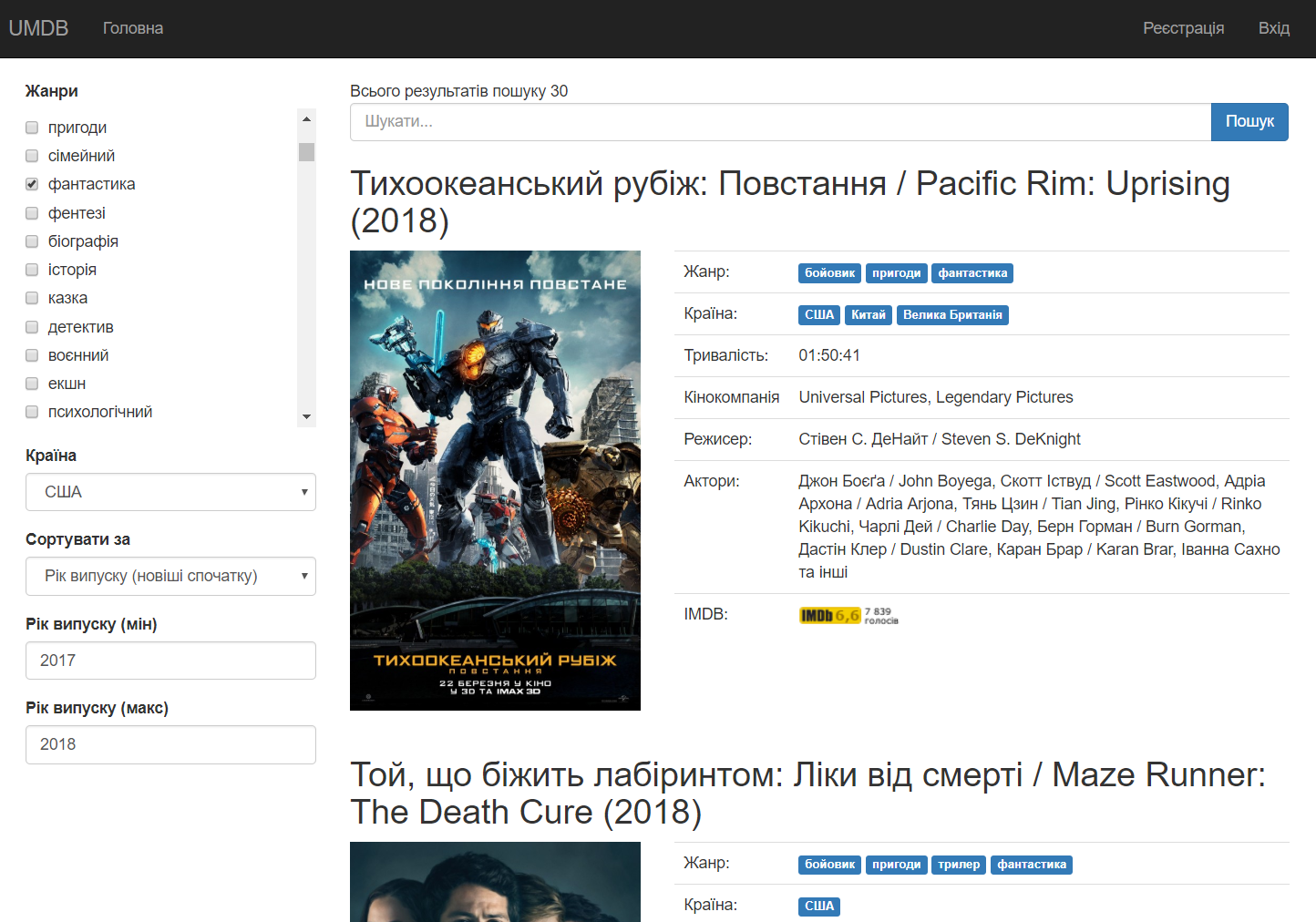
Отож завдяки такому підходу успішно виводяться всі жанри, список країн (у випадаючому списку) і самі фільми, яких на даний момент без застосування жодних фільтрів 8150, як їх і було після розпізнавання парсером. Варто зауважити, що кожен фільм має ярлики з жанрами та країною виробництва, клацнувши на які можна буде здійснювати пошук. Тепер необхідно реалізувати функцію пошуку за всіма критеріями.

3.6 Функції пошуку

Реалізація підключення до бази даних виконана з використаннями Entity Framework Core тому для реалізації запитів до бази даних будуть використовуватися Linq запити. Деякі поля, як от наприклад жанри, країна, чи назва можуть бути не задані, тому були прийнято рішення спочатку виконувати запит на отримання всіх фільмів, а потім у відповідності до наявності того чи іншого параметра виконувати фільтрацію. Наприклад, якщо пошук тільки за назвою, то фільтрація буде виконуватися тільки за назвою, решта буде ігноруватися, а якщо задано пошук ще й за жанром, то окрім пошуку за назвою буде також здійснюватися пошук за жанром. Усе це вдалося реалізувати завдяки використанню умовного оператора if(). Якщо певний параметр пошуку не порожній то за ним здійснюється фільтрація усіх фільмів, інакше він ігнорується. Цей підхід не буде давати велике навантаження на програму оскільки Entity Framework Core кешує інформацію з бази даних.

Для обмеження кількості фільмів, які виводяться на одну сторінку скористаюся методами Skip(int count) та Take(int count), перший метод пропускає певну кількість елементів, другий захоплює певну кількість елементів. Метод Take буде приймати задану константу в 20 елементів на сторінку. Для методу Skip задам формулу для обчислення кількості елементів, які необхідно пропустити для певної сторінки.

Варто зауважити, що всі запити будуть йти за допомогою методу GET оскільки немає потреби приховувати параметри запиту та для зручності поділу на сторінки. Проведу тестування пошуку вивівши всі фільми з 2017 по 2018 рік в яких жанром є фантастика та які були зняті в США.



***Рисунок 3.5*** Результати пошуку

Як бачимо пошук працює справно відображено 30 фільмів за заданим критерієм, посортовані за роком випуску(новіші спочатку). Аналогічно пошук працює і при заданні також імені, причому навіть не знаючи повної назви фільму можна здійснювати пошук. Також назву для пошуку можна задавати як українську так і оригінальну.

3.7 Реєстрація користувачів та вхід

Реєстрація та вхід користувачів були також реалізовані з використанням відповідних моделей, а дизайн з використанням бібліотеки BootStrap 3. Обидві моделі для логінування та реєстрації мають перевірку на заповнення всіх полів і у випадку незадання чи неправильного форматування деякого поля користувач отримає повідомлення про помилку. В якості системи для зв’язку між клієнтом та сервером було використано технологію Identity, тому що вона містить зручну та швидку реалізацію комунікації між сервером та клієнтом, а також дає змогу захищати аккаунт спеціальними ідентифікаторами та хешем паролю.

3.8 Відгуки до фільмів

Для сторінки перегляду кожного фільму окрім детальнішої інформації було також розроблено можливість залишати відгук зареєстрованим користувачам, як це продемонстровано на ***Рисунку 3.6.***



***Рисунок 3.6 Відгуки до фільму та форма для їх додавання***

# Висновки

Отож для створення централізованої бази даних фільмів українською мовою, в якій українці мали б змогу шукати та обговорювати фільми довелося скористатися технологією ASP .NET Core MVC, яка дає зручну можливість реалізувати такий складний проект. Також довелося розібратися у всіх тонкощах взаємодії між сервером та клієнтом аби створити програму-парсер для заповнення бази фільмами. Для самої програми-парсеру було використано чималу кількість парсингу DOM структури html сторінок та регулярних виразів, без яких розпізнавання такої кількості інформації було б дуже важким та тривалим.

В перспективі розвитку проекту є додавання власної системи рейтингу фільмів, списку з 100 найбільш популярних фільмів за рейтингом, списку схожих фільмів, списку улюблених фільмів для кожного з користувачів, а також поділ фільмів на певні категорії(сімейні, молодіжні, постапокаліптичні і т.д.). В планах також допрацювати систему парсеру, аби додавання нових фільмів здійснювалося автоматично через певний проміжок часу.

# Список використаних джерел

1. Adam Freeman Pro ASP.NET Core MVC 2 7th ed. Edition, London, UK, 2017. – 1017 с.
2. Adam Freeman Pro Entity Framework Core 2 for ASP.NET Core MVC, London, UK, 2017. – 650 c.
3. Andrew Troelsen Pro C# 7: With .NET and .NET Core, Minneapolis, Minnesota, USA, 2017. – 1372 c.
4. <http://html-agility-pack.net/documentation>
5. <https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура>
6. <https://en.wikipedia.org/wiki/Client–server_model>
7. <https://uk.wikipedia.org/wiki/Куки>
8. <https://en.wikipedia.org/wiki/HTTP_cookie>
9. <https://stackoverflow.com/questions/8490533/notify-observablecollection-when-item-changes>
10. <https://stackoverflow.com/questions/13049732/automatically-rename-a-file-if-it-already-exists-in-windows-way>
11. <https://msdn.microsoft.com/en-us/library/ms178194.aspx>
12. <http://helpyourselfhere.blogspot.com/2015/02/wpf-mvvm-relaycommand-implementation.html>
13. <https://stackoverflow.com/questions/16684370/how-to-get-listview-itemclick-vallue-in-relay-command>
14. <https://stackoverflow.com/questions/3615800/download-image-from-the-site-in-net-c>