**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ІВАНА ФРАНКА**

Факультет прикладної математики та інформатики

(повне найменування назва факультету)

Дискретного аналізу та інформаційних систем

(повна назва кафедри)

**КУРСОВА РОБОТА**

на тему:

Розробка web-орієнтованої системи «Українська база даних фільмів – Ukrainian Movie DataBase(UMBD)»

Студента 3 курсу, групи ПМІ-33 ,

напряму підготовки інформатика

Романюка Б.І.

(прізвище та ініціали)

Керівник ?????? Квасниця Г.А.

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

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

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

Львів – 2018

Зміст

[Зміст 2](#_Toc515287572)

[I. Вступ 3](#_Toc515287573)

[II. Постановка задачі 3](#_Toc515287574)

[III. Актуальність тематики 4](#_Toc515287575)

[IV. Побудова парсеру та парсинг інформації 4](#_Toc515287576)

[4.1 Дослідження та побудова моделі 4](#_Toc515287577)

[4.2 Побудова графічного інтерфейсу парсеру та підключення необхідних бібліотек для парсингу інформації 6](#_Toc515287578)

[4.3 Логінування програми під виглядом звичайного користувача 7](#_Toc515287579)

[4.3.1 Дослідження взаємодії між клієнтом та сервером 7](#_Toc515287580)

[4.3.2 Створення функціональності для зберігання cookie файлів та здійснення GET запитів з їх допомогою 8](#_Toc515287581)

[4.4 Парсинг фільмів 9](#_Toc515287582)

[4.5 Розпізнавання жанру та країни виробництва для окремого фільму 15](#_Toc515287583)

[4.6 Зберігання інформації в бінарному форматі 16](#_Toc515287584)

[4.7 Завантаження постерів 17](#_Toc515287585)

[4.8 Оновлення бази фільмів 19](#_Toc515287586)

[4.9 Пошук за різними критеріями 20](#_Toc515287587)

[4.10 Розпізнавання та зберігання всіх фільмів та постерів. 21](#_Toc515287588)

[V. Побудова web-орієнтованої системи 22](#_Toc515287589)

# Вступ

Метою даної курсової роботи є створення веб-орієнтованої системи, яка б містила велику кількість фільмів та інформації про них і вся інформація були при цьому українською мовою. Для створення самого сайту буде використано технологію ASP .NET Core з використанням патерну проектування MVC. Сайт буде давати можливість користувачам шукати, реєструватися, логінуватися, обговорювати, рейтенгувати, створювати списки схожих фільмів та багато іншого функціоналу і все це виключно українською мовою. Інформація про всі фільми буде зберігатися у базі даних, в якій таблиця фільмів містити 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. А ще кожен фільм міститиме список схожих фільмів, і додавати схожі фільми також зможуть зареєстровані користувачі. Тобто значну роботу із організації заповнення сайту буде покладено на самих користувачів.

Проте 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 розпарсити 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 Cookies 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.

# Постановка задачі

Створити централізовану базу даних фільмів українською мовою, в якій українці мали б змогу шукати фільми, обговорювати їх та ставити рейтинг.  
До особливих функцій сайту можна віднести функцію розширеного пошуку, яка буде давати наступні можливості:  
-фільтрування за жанром, роками, країною випуску

-фільтрування за режисером, акторським складом, кіностудіями, кінокомпаніями

-фільтрування за тривалістю  
-фільтрування за категорією(постапокаліпчичні фільми, молодіжні фільми і тд), причому категорії можуть створювати самі користувачі  
-топ 250 фільмів за рейтингом користувачів

-список схожих фільмів

# Актуальність тематики

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.

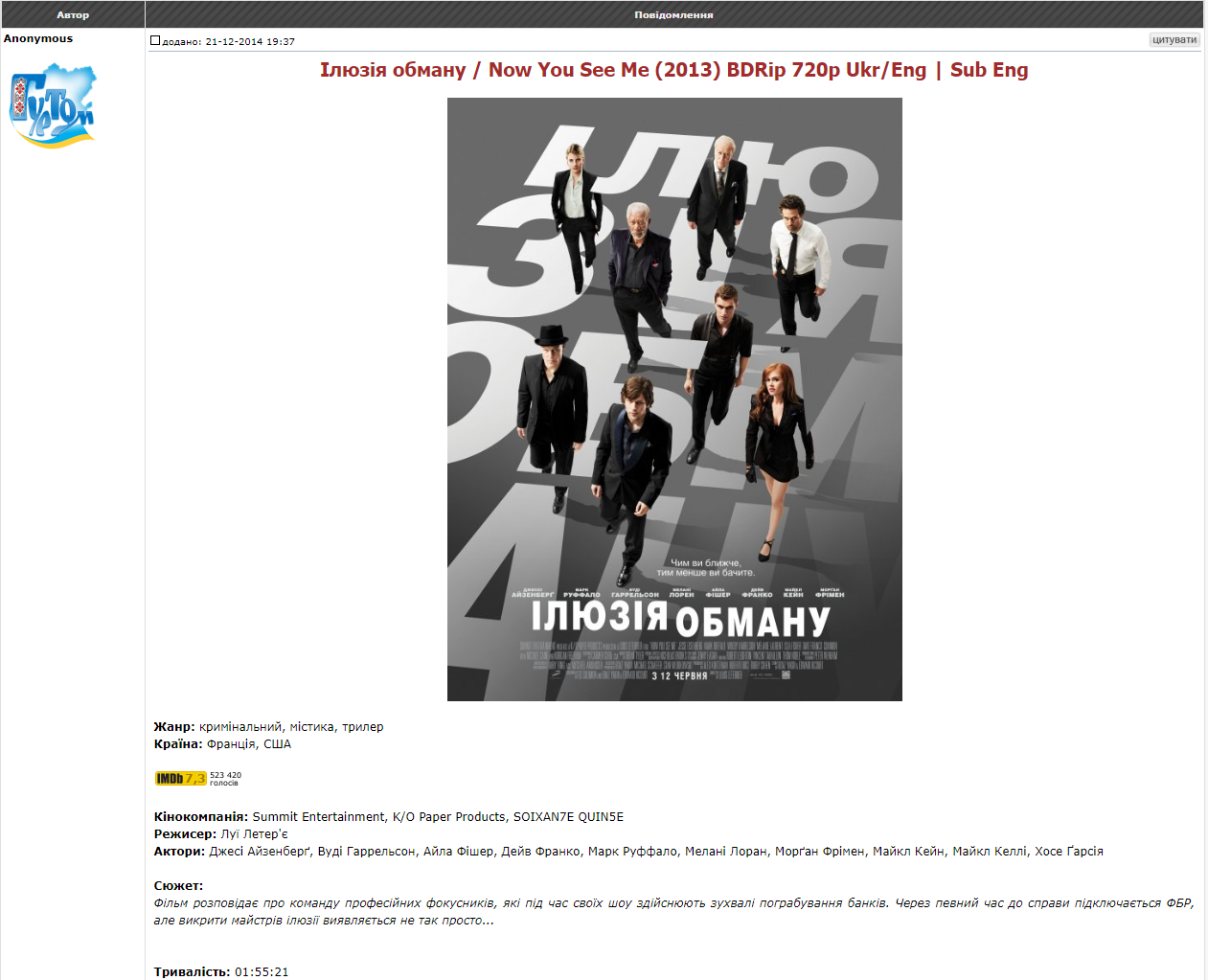
Розглянемо кілька прикладів:

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. Також сайт не дає змоги користувачам додавати схожі фільми.
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 рейтенгувати фільм будучи не зареєстрованим користувачем, а також можливість залишати відгуки зареєстрованим користувачам.
3. Український сайт **hurtom.com** є найбільш прибавливим у плані обговорення фільмів та рейтенгування. 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.

# Побудова парсеру та парсинг інформації

4.1 Дослідження та побудова моделі

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 4-1***. Також варто зауважити, що всі фільми зберігаються в окремому розділі, поділеному на сторінки по 45 фільмів на сторінку.



***Рисунок 4-1.*** 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

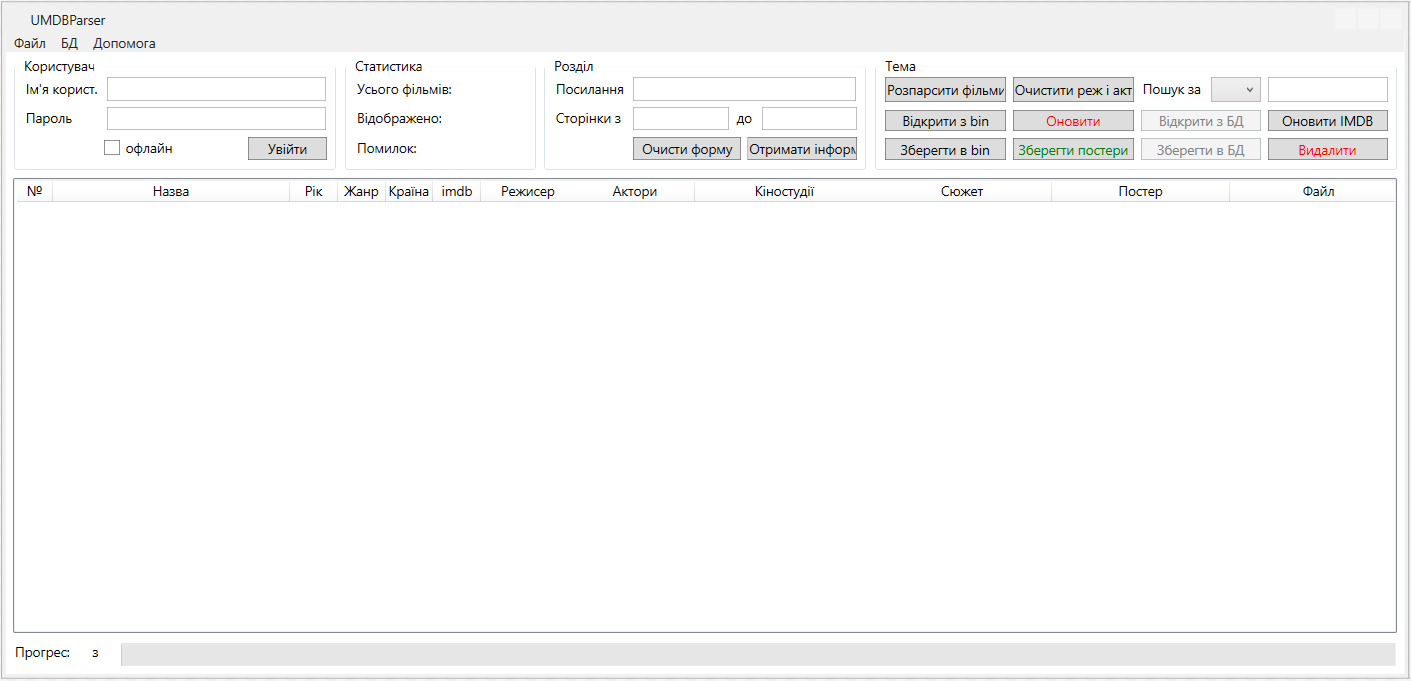
Отож згідно даної стурукрутри модель фільму міститиме наступні поля:

* Назва фільму (українською мовою та оригінальна назва)
* 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

4.2 Побудова графічного інтерфейсу парсеру та підключення необхідних бібліотек для парсингу інформації

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 4.1*** 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.



***Рисунку 4.1*** Основне вікно програми-парсера

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

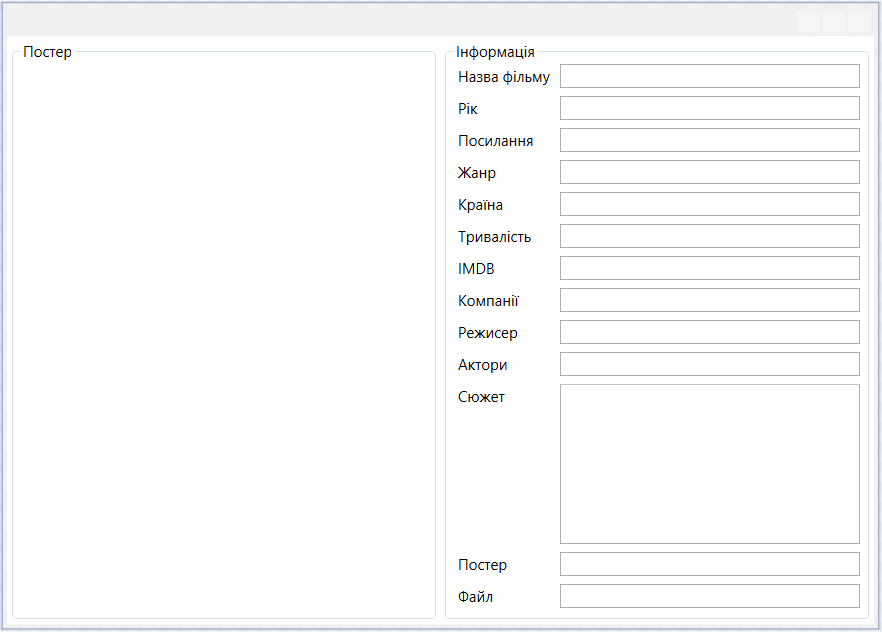
Меню містить основні функції програми, такі як зберігання та відкривання уже розпізнаних фільмів, збереження та оновлення інформації у базі даних.

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. контроли для вказування посилання на розділ з якого буде братися інформація та діапазон сторінок
4. контроли для парсингу окремо кожного фільму, зберігання всієї інформації у бінарному формат та її відкривання, оновлення поточного списку новими фільмами, збереження постерів, функція пошуку, оновлення інформації з сайту imdb.com 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 окремо також створив окреме вікно, яке буде відкриватися при клацанні на відповідний фільм у списку.



***Рисунку 4.2*** 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.

Також аби 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.

4.3 Логінування програми під виглядом звичайного користувача

4.3.1 Дослідження взаємодії між клієнтом та сервером

Як уже згадувалося раніше перегляд значної частини фільмів неможливий без реєстрації, навіть якщо є посилання на фільм 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 веб-сервером для ідентифікації користувача. Якщо користувач увійшов під своїм іменем на певному сайті то веб-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 застрілі cookie вже не зможе користуватися сайтом. 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, а користувач помітив підозрілу активність, він може просто дати запит на знешкодження файлів cookie (вихід з профілю на сайті) як наслідок зловмисник теж втратить доступ до профілю.

4.3.2 Створення функціональності для зберігання cookie файлів та здійснення GET запитів з їх допомогою

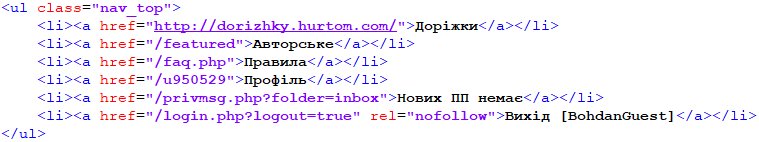
HtmlAgilityPack уже містить функціональність для завантаження html сторінок та задання спеціального контейнера CookieCollection, з допомогою якого потім будуть здійснюватися запити, проте задання кожного ключа і значення є клопітким процесом, а ще через деякий час кукі втрачають свою валідність і сервер перестане сприймати запити від програми. Тому було вирішено створити клас 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 відправки GET та POST запитів з використанням cookie, які повертатимуть об’єкт HtmlDocumment з цієї ж бібліотеки за допомогою якого і буде здійснюватися пошук необхідної інформації. В даній реалізації під час надсилання POST запиту контейнер з кукі файлами буде встановлюватися і зберігатися, а при надсиланні GET запиту тільки використовуватися.

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 4.2 та 4.3 це дійсно так і сервер сприймає програму парсер як звичайного користувача, що увійшов в систему.

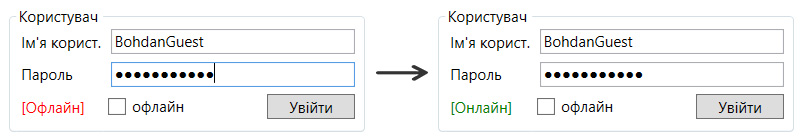


***Рисунок 4.3*** Html код без використання cookie (наявні Зареєстуватися та Вхід)



***Рисунок 4.4*** 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.



***Рисунку 4.5*** Зміна ідентифікатора під час входу користувача

Тепер можна приступати до парсингу самих фільмів.

4.4 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).

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

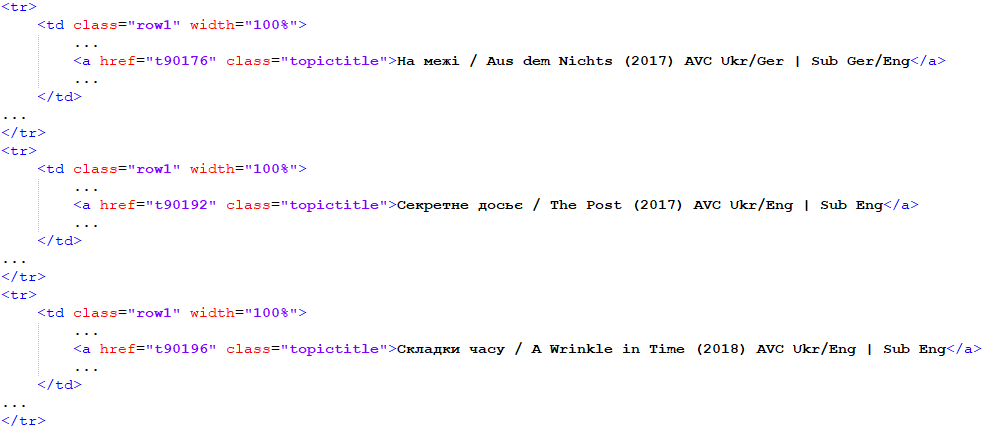
<https://toloka.to/f70-90> 3-тя сторінка

…………

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"**



***Рисунку 4.6*** 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 регулярними виразами, адже рік знаходиться в дужках.

Regex.Match(name, @"(\([0-9]{4}\))"), де name – повна назва з роком

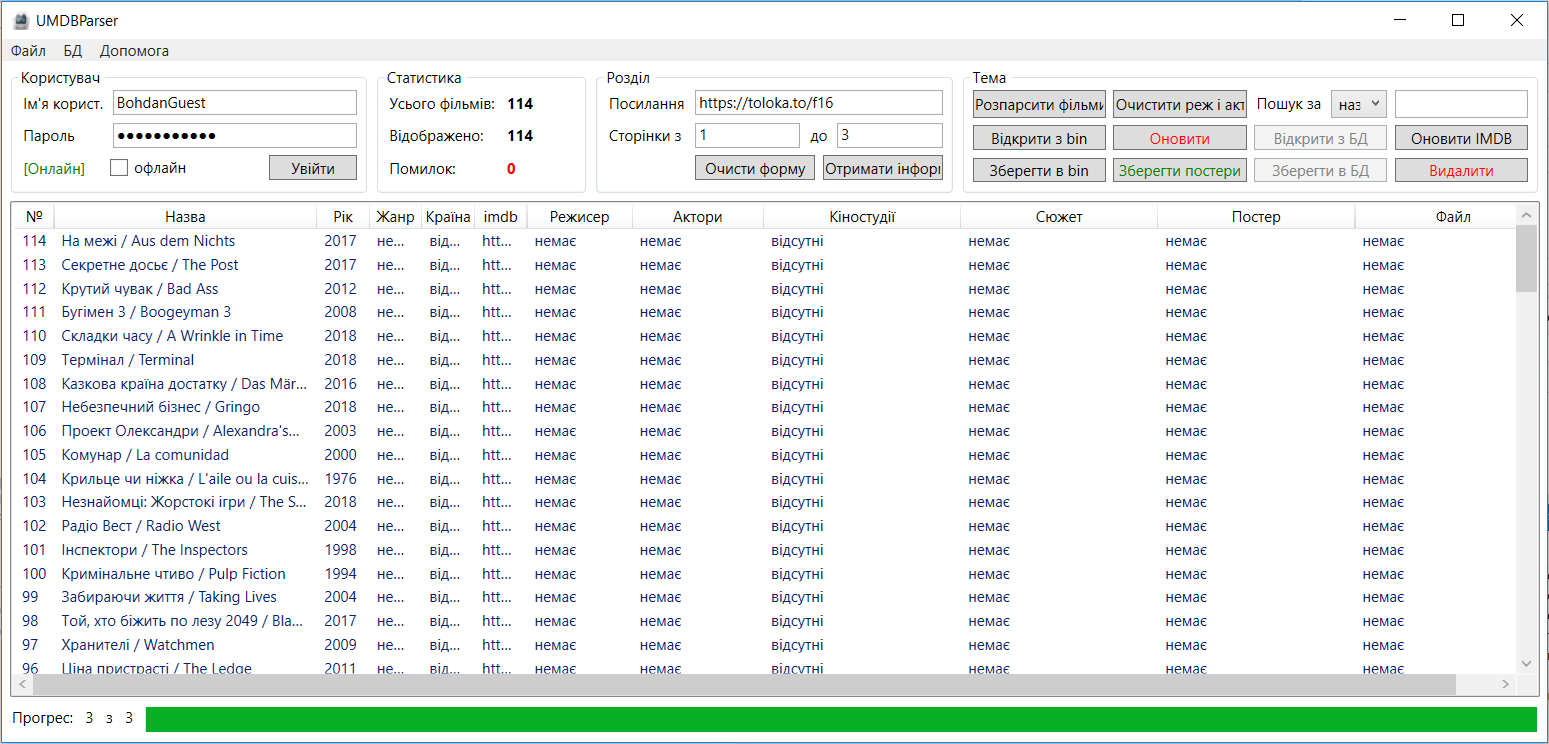
Даний регулярний вираз шукає чотири повторення цифр від 0 до 9 записаних у дужках. Весь текст до цього – це назва фільму(українська + оригінальна), а отриманий результат регулярного виразу – рік. Також вважається, що назва фільму унікальна, а повтори у результат не включаються.

Варто зауважити, що оскільки завантаження сторінки та її обробка займаються 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, аби розподілити отримання сторінки та її розпізнавання на два окремих процеси, що дало змогу вільно переглядати процес виконання та уникнути неактивності графічного інтерфейсу. В даному випадку метод для отримання інформації GetAllInfo 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-оьх сторінок

Отож як видно з **Рисунка 4.5** 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 модель, але не відображається на графічному інтерфейсі головного вікна, його можна переглянути лише під час перегляду детальнішої інформації.

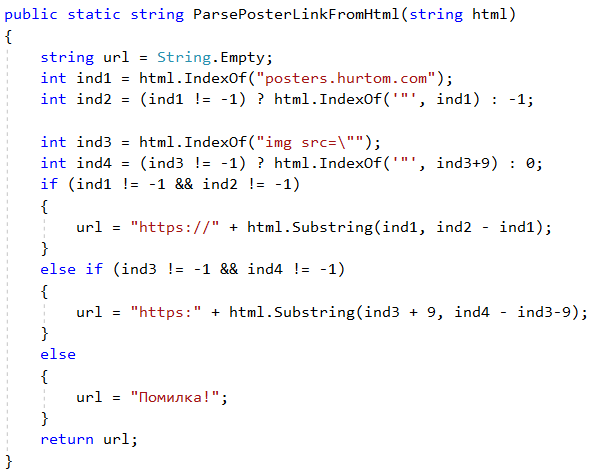


***Рисунку 4.7*** Перший етап парсингу фільмів

На другому етапі будую конкатенацію посилання <https://toloka.to> та отриманого на першому етапі посилання і завантажую сторінку з детальною інформацією. З html коду кожної теми з фільмом можна побачити, що всі повідомлення знаходяться в середині елементів **<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.

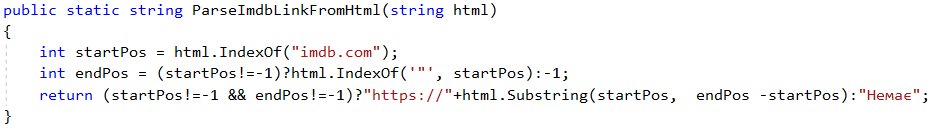
Спершу знайду посилання на постер до фільму, для цього створю статичний метод **ParsePosterLinkFromHtml** у класі **Movie**, який буде шукати перше входження посилання на дочірній сайт posters.hurtom.com або якщо такого не має то першу картинку, яка зустрічається



***Рисунок 4.8*** Пошук постеру

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 ':','\*','?', '"', '<', '>' то заміню їх в назві на пробіл , а символ '/' на '-' і додавши до назви рік та розширення файлу отримаю назву постеру.

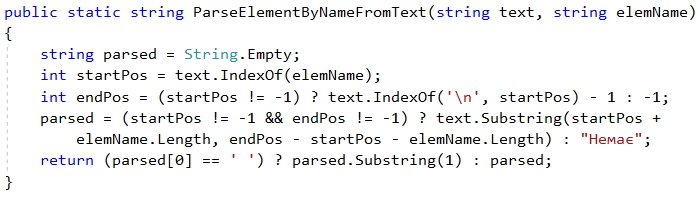
Далі також необхідно знайти посилання на сайт **imdb.com**, для цього створю інший статичний метод **ParseImdbLinkFromHtml** у класі Movie з наступним кодом.



***Рисунок 4.9*** Метод ParseImdbLinkFromHtml

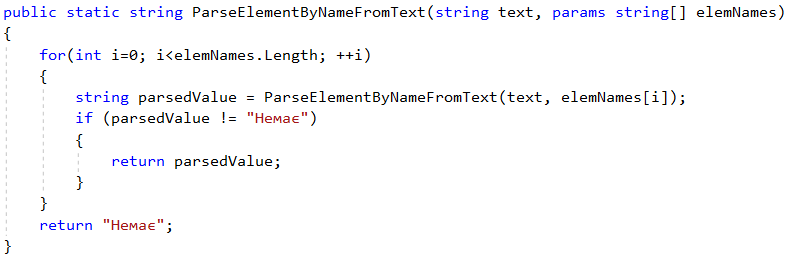
Алгоритм шукає перше входження рядка imdb.com, якщо таке існує і повертає повне посилання або інформацію про те що його немає.

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.



***Рисунок 4.10*** Метод ParseElementByNameFromText

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



***Рисунок 4.11*** Перевантаження методу ParseElementByNameFromText

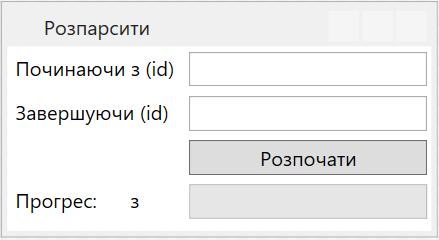
Для пошуку сюжету всього вище перерахованого буде недостатньо, адже опис сюжету починається з наступного рядочку після слова «Сюжет», а попередні алгоритми розраховані лише на випадки коли інформація знаходиться у тому ж рядочку**.** Щоб отримати сюжет фільму доведеться врахувати той факт що сюжет розтягується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

Regex.Match(firstPost, "<div style=\"text-align: justify;\"><span style=\"font-style: italic\">(.|\n)\*?</span></div>");

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

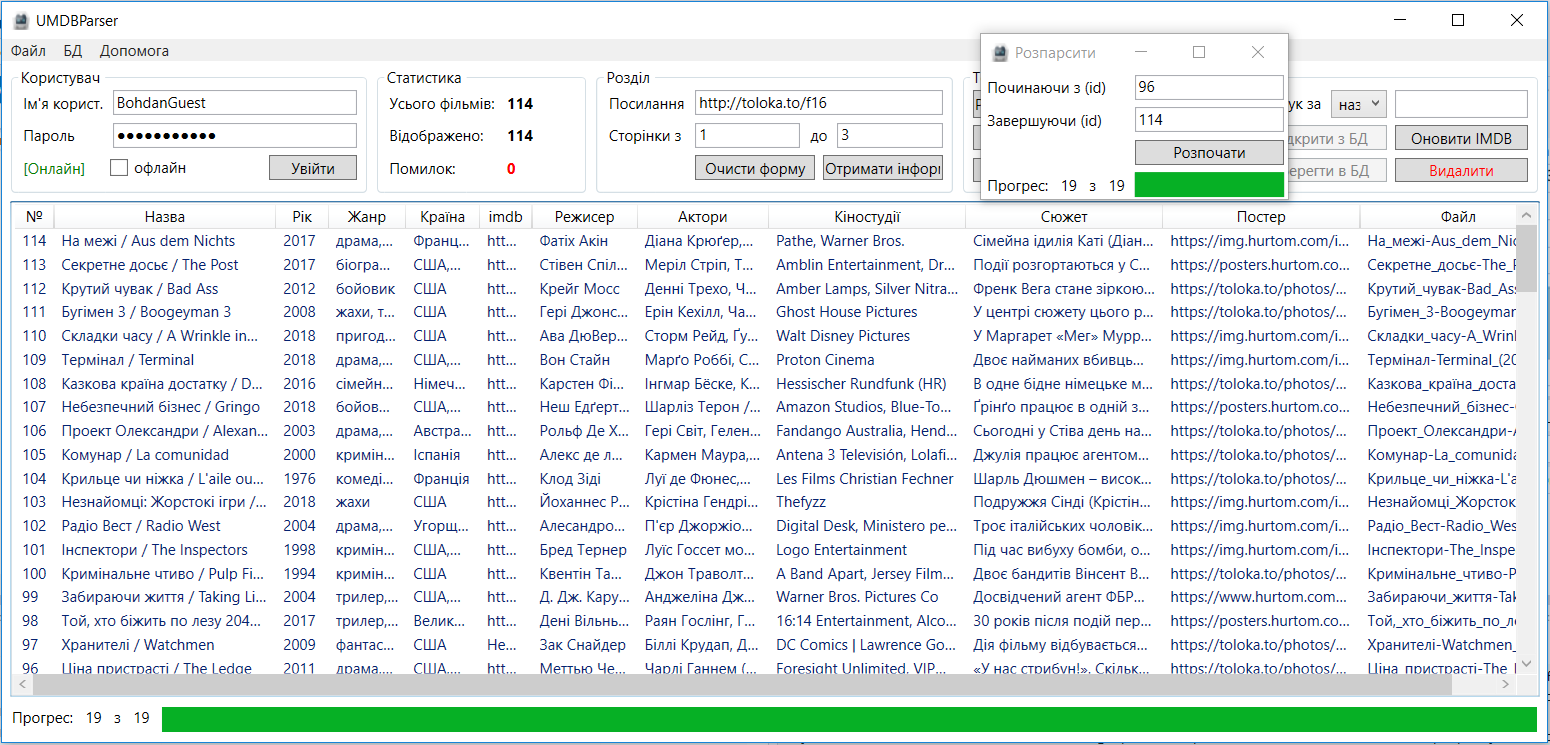
Як у випадку із методом GetAllInfo для отримання інформації на першому етапі частина коду у поточному методі ParseMovies також було позначено за допомогою await блоку, а сам метод специфікатором async. Це також дало змогу розблокувати інтерфейс користувача на час отримання інформації та можливість переглядати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

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



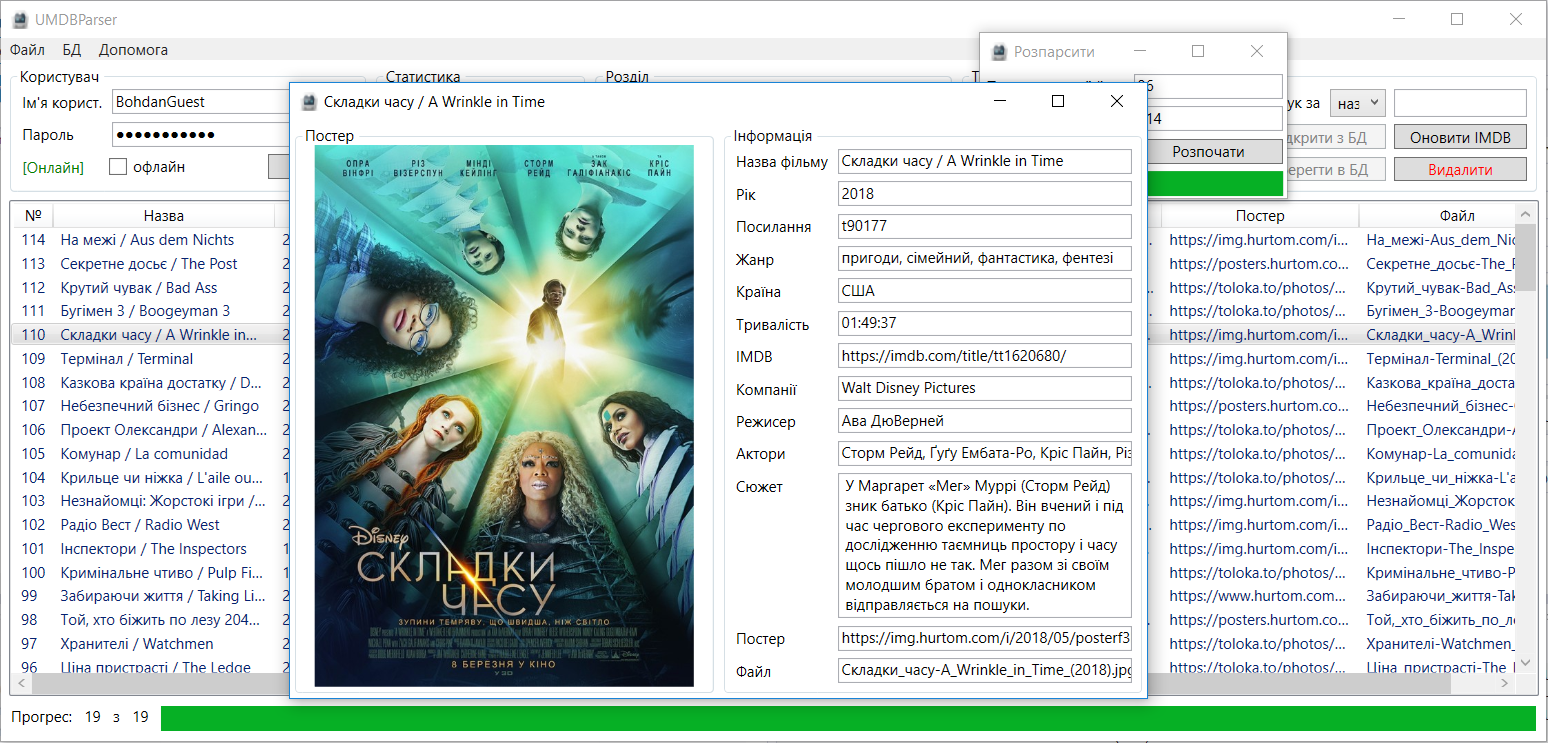
***Рисунок 4.12*** Вікно для парсингу фільмів з певного проміжку

Протестую другий етап отримання інформації розпізнавши фільми з 96 по 114



***Рисунку 4.13*** 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. Також завдяки бібліотечним методам **BitMap** класу підвантажується постер до фільму з посилання.



***Рисунку 4.14*** Перегляд детальної інформації про фільм

4.5 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 у кожному фільмі 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. Спочатку 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.

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.

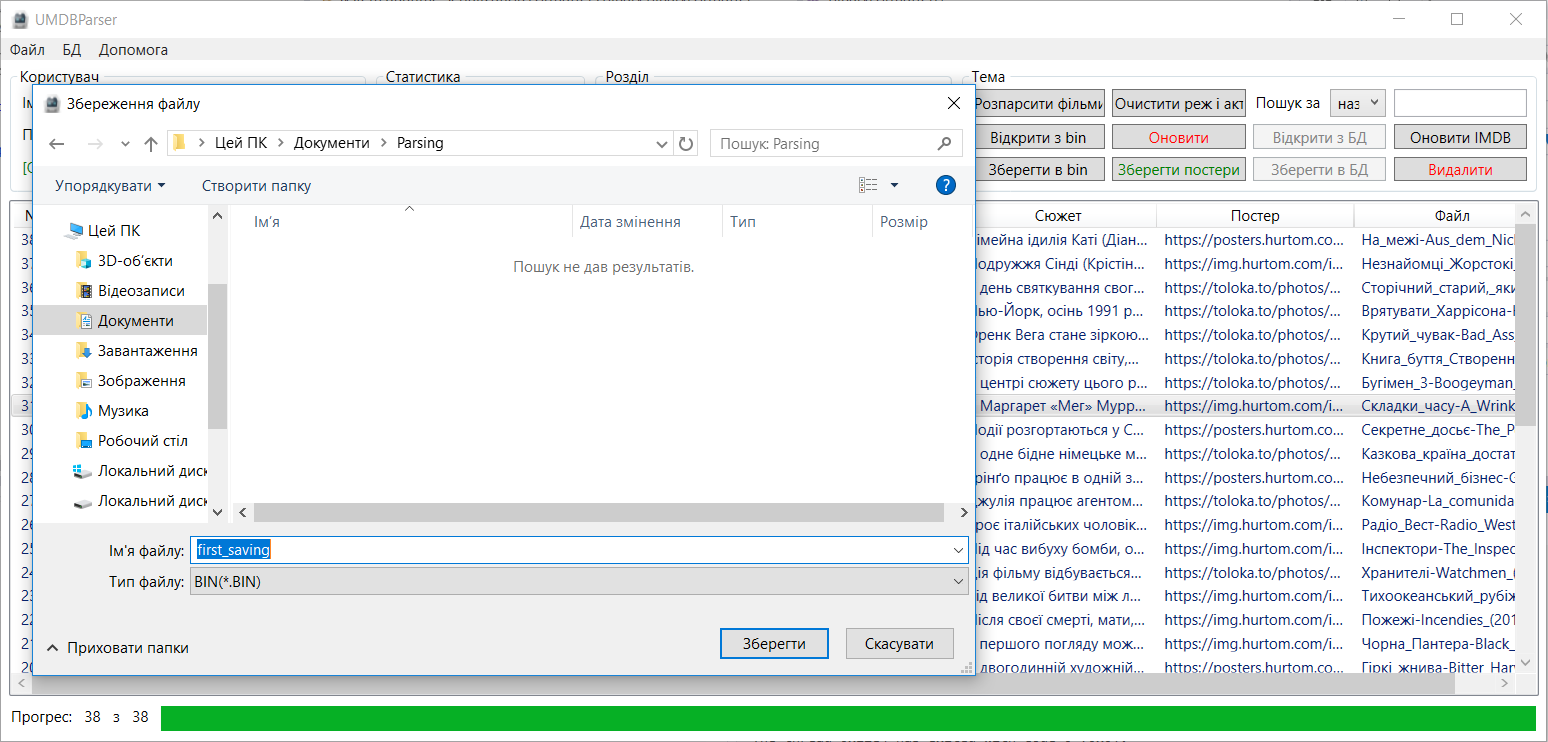
Таким чином перебираючи усі фільми одночасно генеруються словники жанрів та країн і додаються відповідні жанри та країни до кожного фільму окремо.

4.6 Зберігання інформації в бінарному форматі

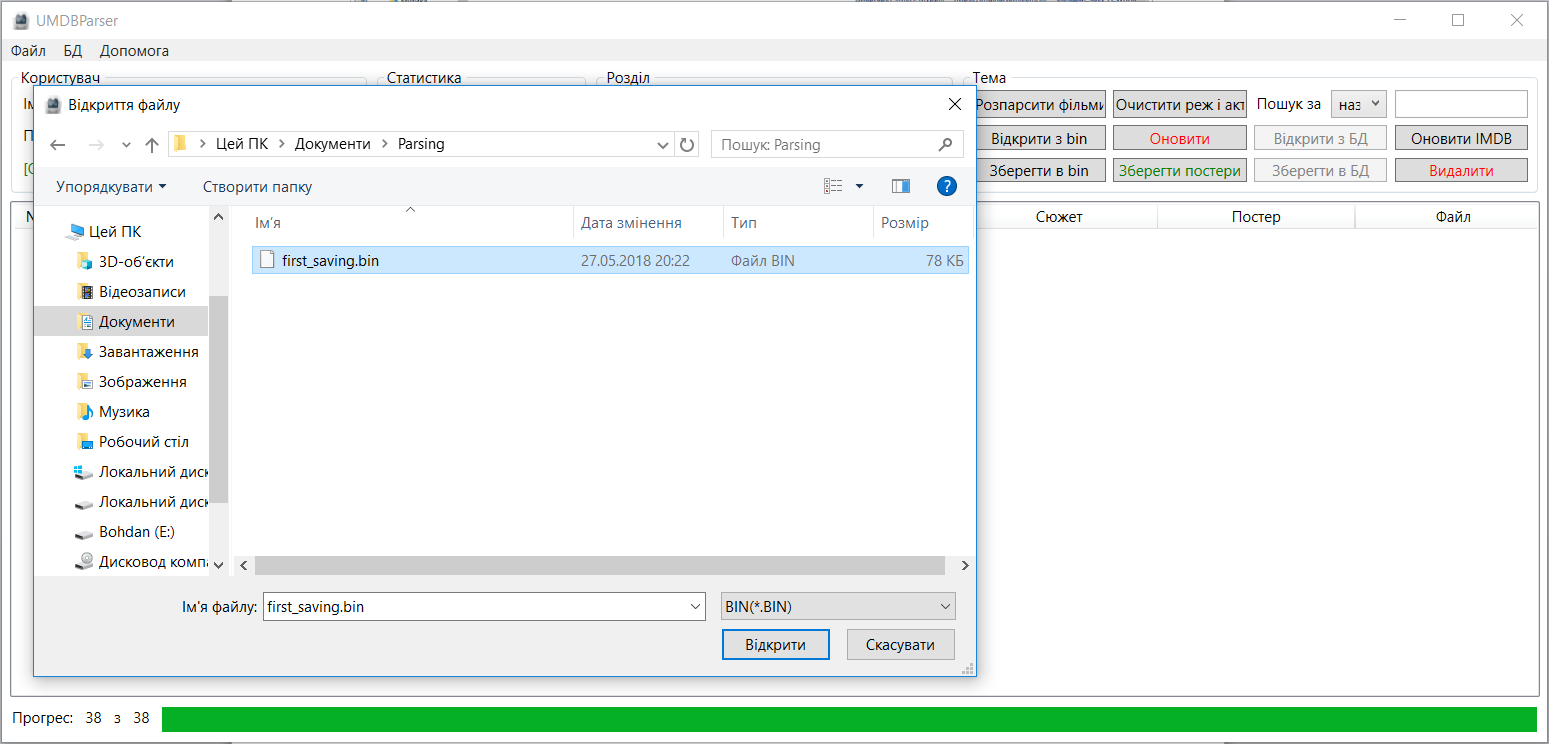
Оскільки остаточне розпізнавання всіх фільмів буде здійснюватися частинами, а не все одразу, аби не перевантажувати сервер з якого беремо інформацію було вирішено тимчасово зберігати дані у бінарному форматі. Бінарний формат обрано тому, що він найзручніший для зберігання великого обсягу інформації, 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.

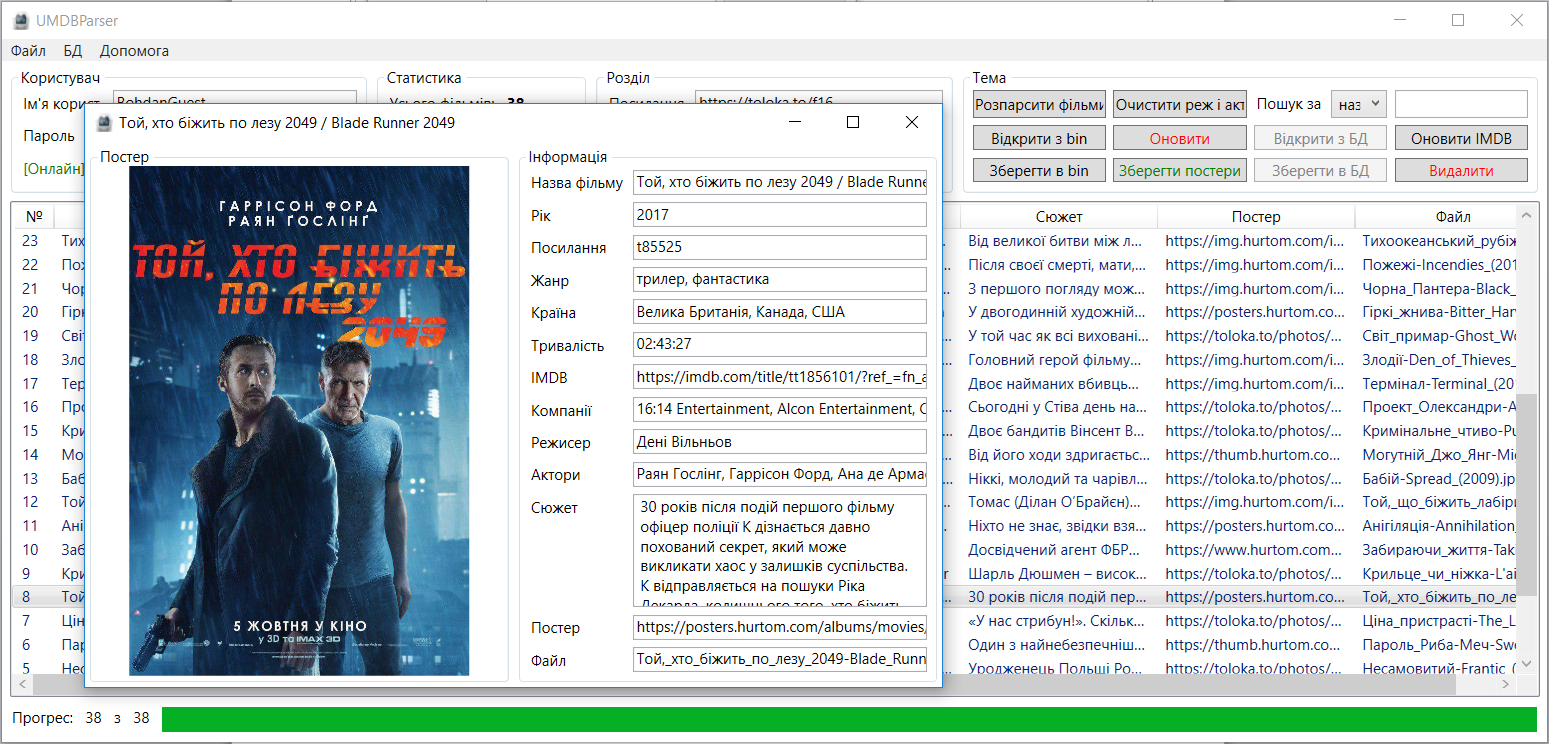
Скориставшись стандартними діалоговими вікнами для зберігання та відкривання файлів проведу перевірку на правильність зберігання у бінарному форматі.



***Рисунку 4.15*** Зберігання у бінарному форматі



***Рисунку 4.16*** Відкриття та десереалізація бінарного файлу



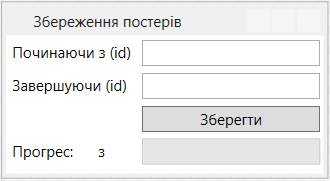
***Рисунку 4.17*** Результат десереалізації бінарного файлу

Отож як видно з ***Рисунка 4.17*** 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.

4.7 Завантаження постерів

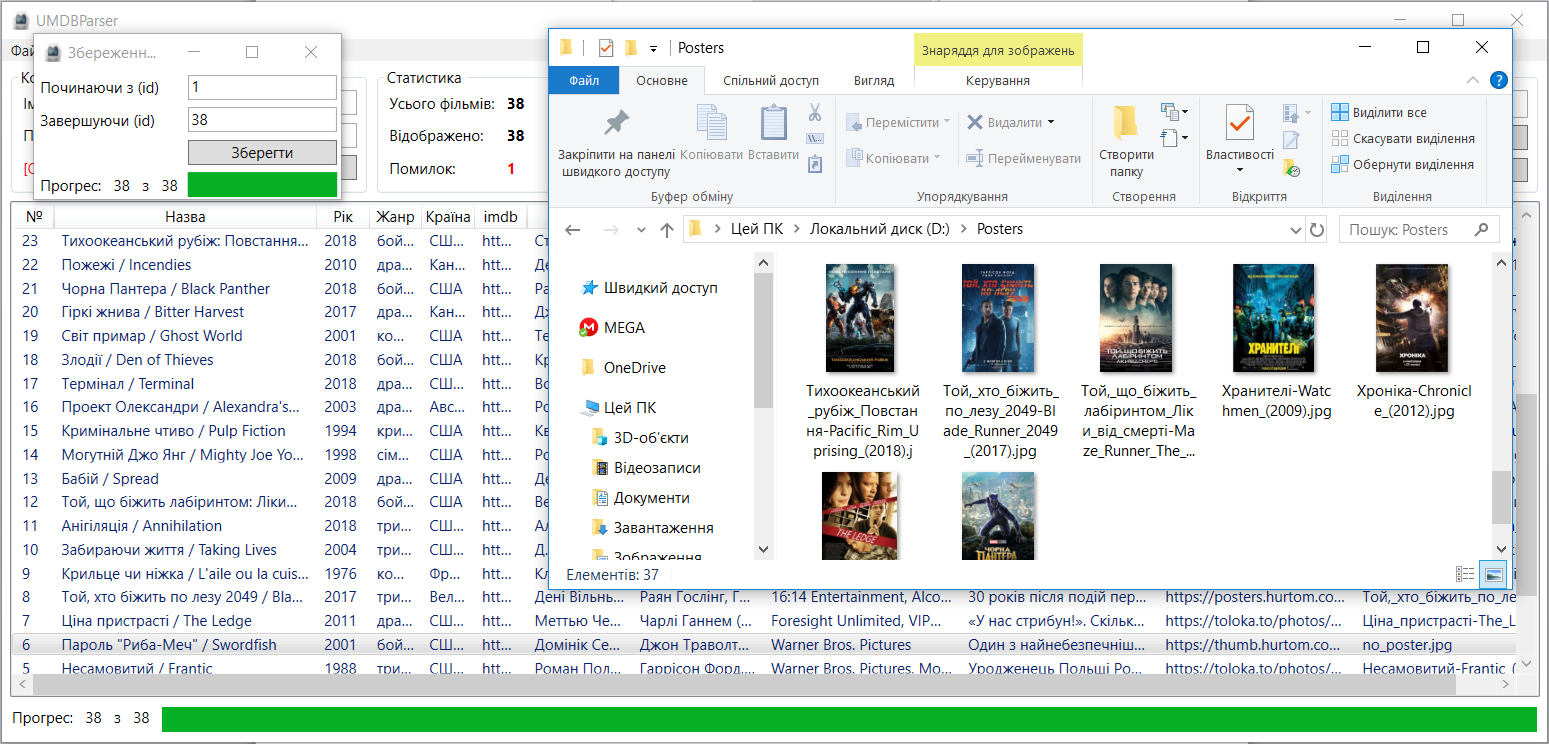
У попередніх пунктах для перегляду окремого постера до кожного фільму потрібно було відкривати детальну інформацію про кожен фільм і програма сама завантажувала постер з посилання і відображала його. Для відображення інформації разом з постерами на веб-сайті необхідно завантажити усі постери у окремий 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 **4.4** 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.



***Рисунку 4.18*** Вікно для збереження постерів з певного проміжку

Також 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.



***Рисунку 4.18*** Результат збереження постерів

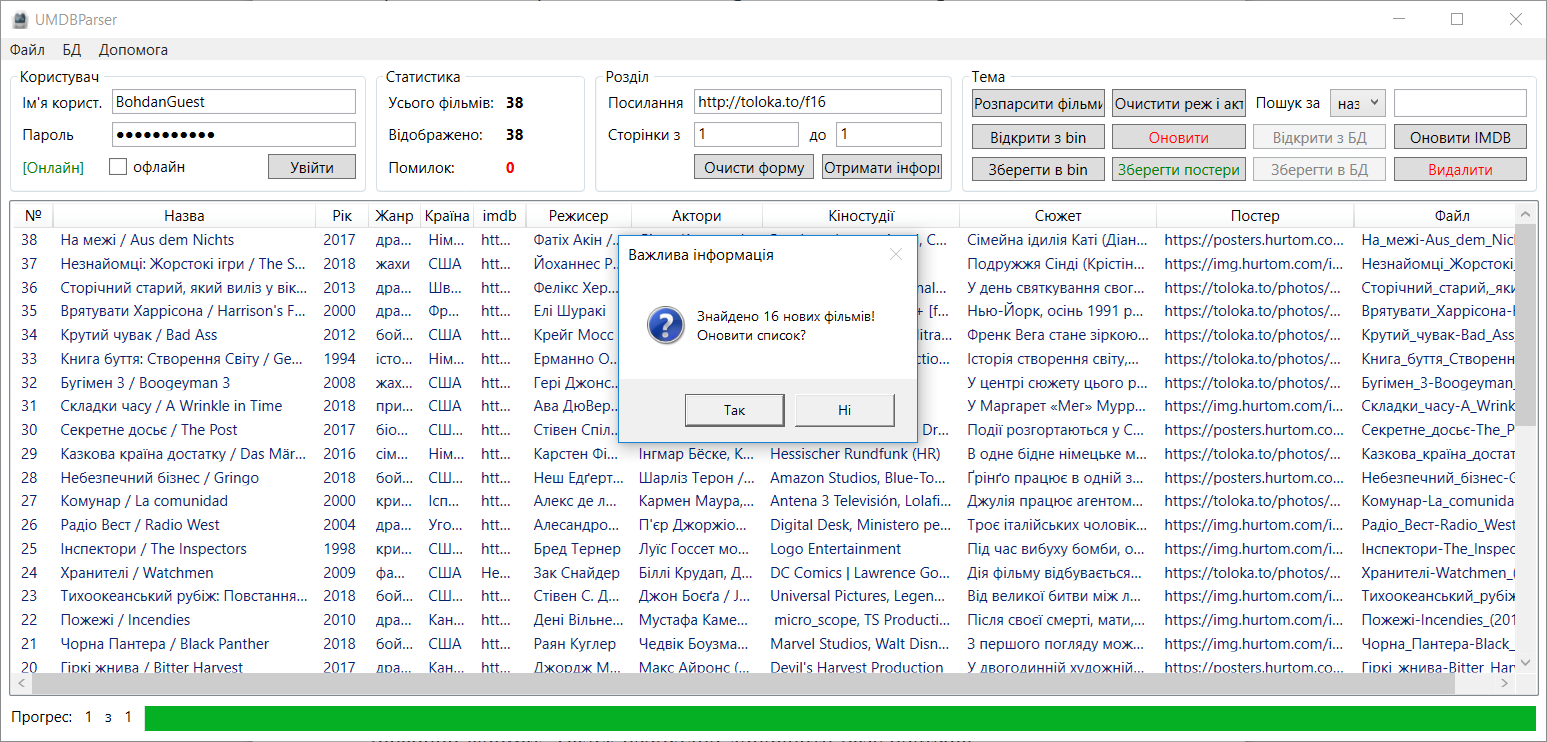
На Рисунку 4.18 з 38 постерів успішно збережені 37 і виникла одна помилка під час збереження постеру до фільму 2001 року, але система успішно замінила назву файлу на no\_poster.jpg.

4.8 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(GetAllInfo), 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. Оскльки можливий випадок коли станется збій в роботі програми нові фільми зберігати в окрему колекцію і лише після підтвердження від користувача про бажання доповнити базу додам цю колекцію до оновної.

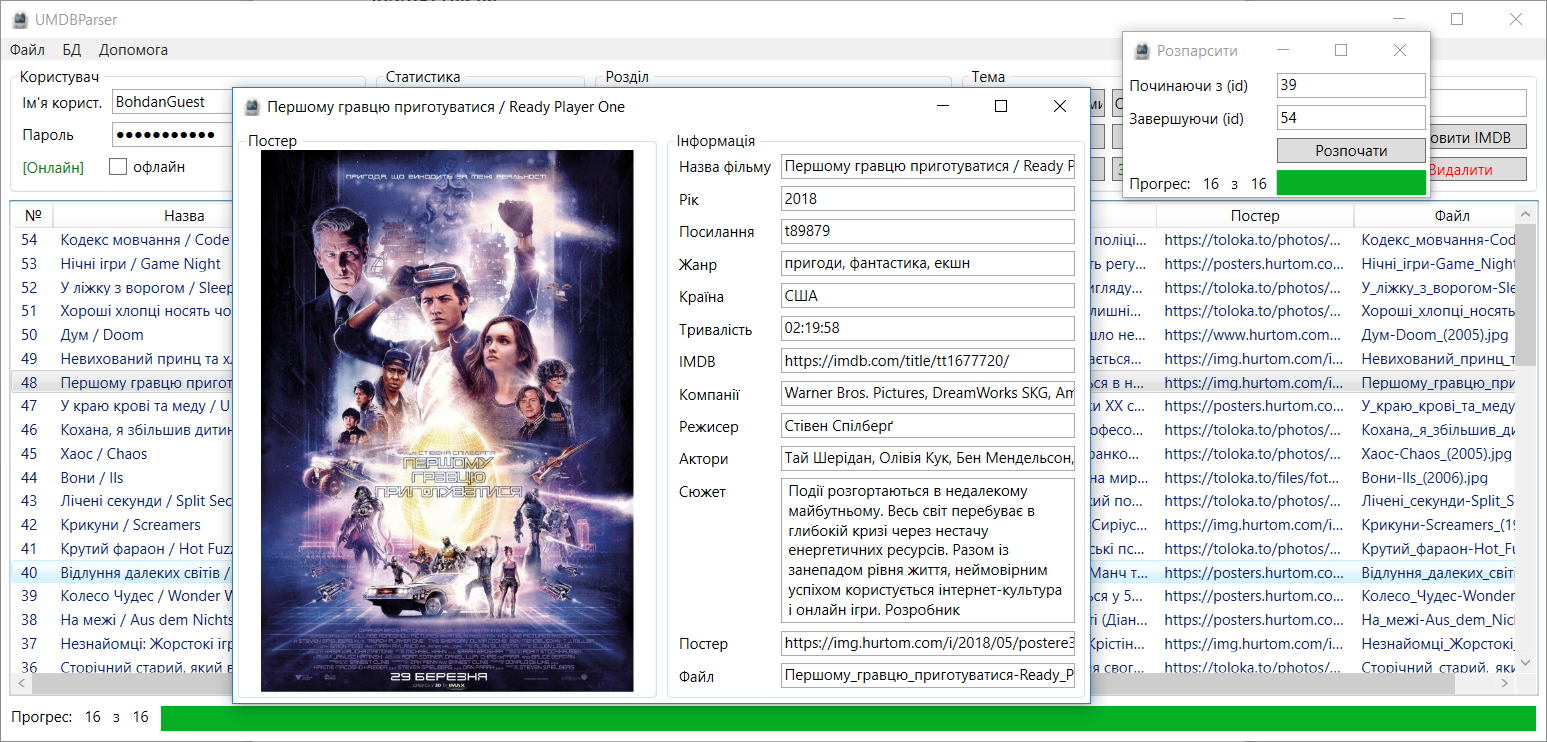
Як і метод GetAllInfo поточний метод також буде виконуватися з використанням багато потоковості аби не блокувати графічний інтерфейс користувача.

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



***Рисунок 4.19*** Доповнення бази даних

Як бачимо методу вдалося знайти 16 нових фільмів, які були додані на сайт впродовж кількох попередніх днів. Давши 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 необхідно доповнити базу постерів.



***Рисунок 4.19*** Отримання всієї інформації про фільми

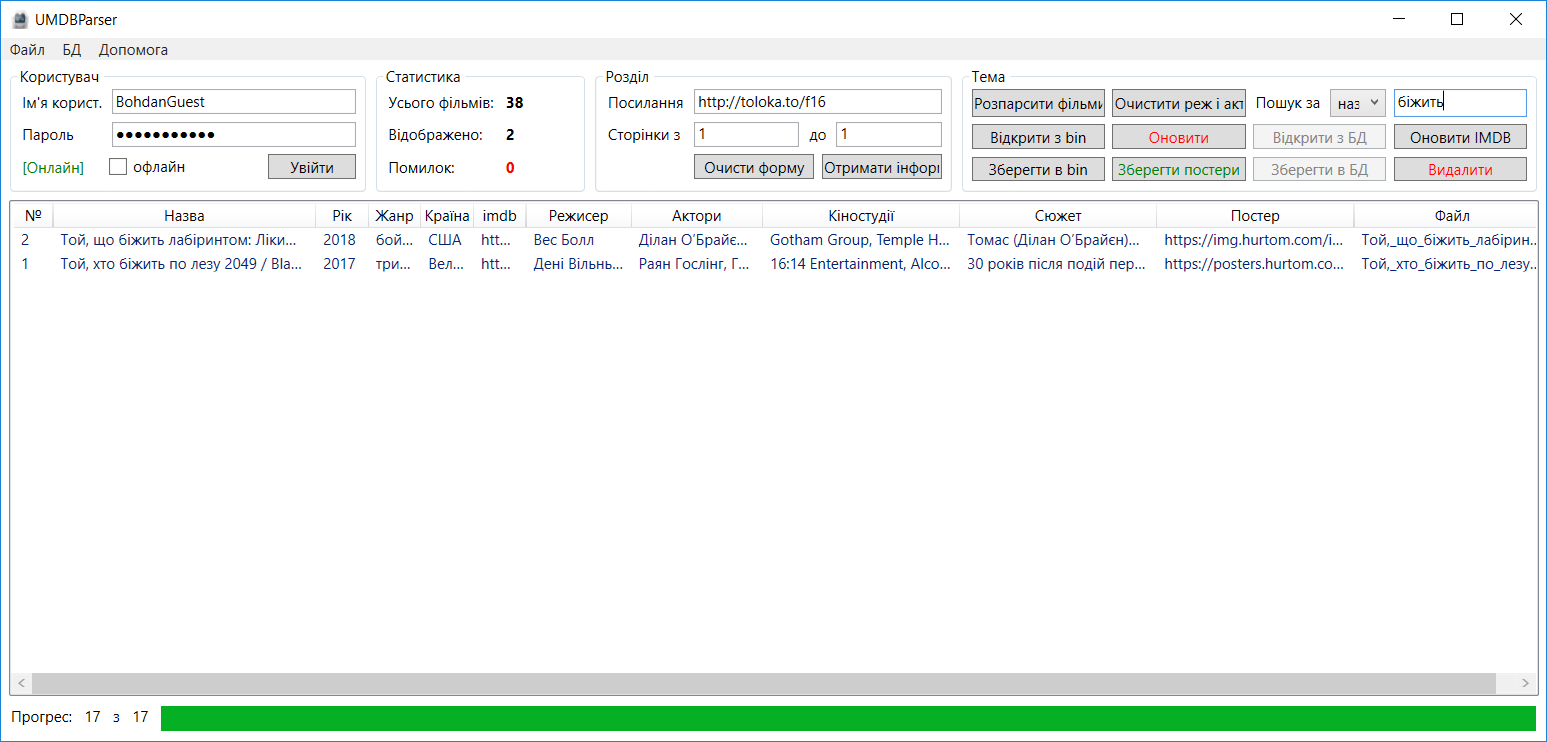
Отож як показано на ***Рисунку 4.19*** метод успішно доповнив базу 16 фільмами і в результаті отримали 54 фільми всього.

4.9 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 4.20*** Пошук за ключовим словом «біжить»

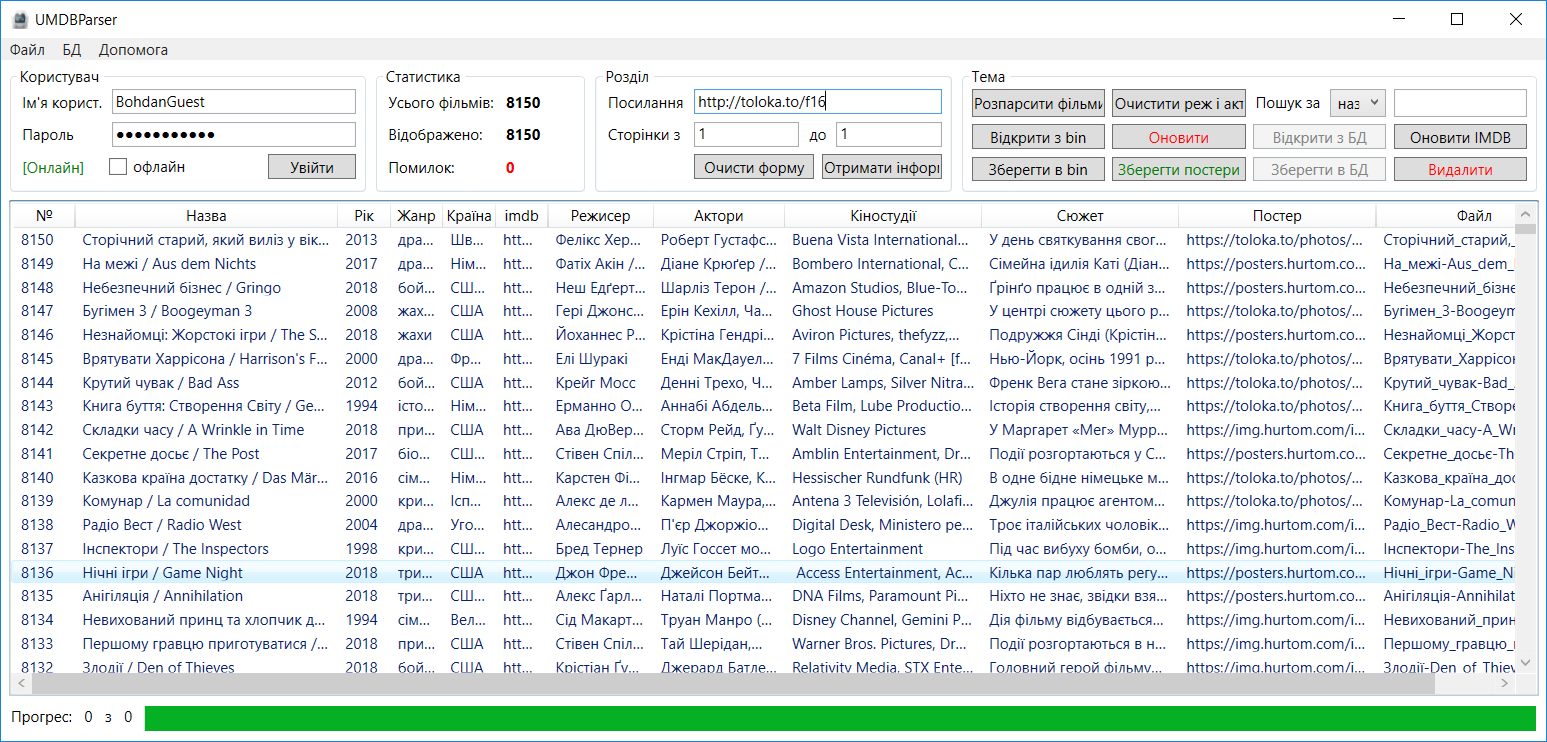
4.10 Розпізнавання та зберігання всіх фільмів та постерів.

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 потратив ~3 годин на розпізнавання всіх 8150 фільмів, як це показано на ***Рисунку 4.21***. Це набагато швидше ніж людина зробила б вручну. Для даного списку фільмів також було згенерованою список жанрів, розмір якого склав 163 жанри та список країн, яких виявилося 135. Після збереження у бінарний файл, його розмір виявився 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 4.21*** Результат загального парсингу усіх фільмів

# Побудова web-орієнтованої системи

5.1 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 користувачі на сайті буде визначатися ще двома табличками:

* 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.

* Таблиця **Ratings**(MovieId, RateValue, RatedById)

Містить ідентифікатор фільму, який було оцінено, оцінку від 0 до 10 та ідентифікатор користувача, який залишив цю оцінку.

5.3 ER-діаграма бази даних

На ***Рисунку 5.1*** зображена ER-діаграми бази даних зі всіма табличками та зв’язками між ними.

***Рисунок 5.1*** ER-діаграма бази даних