Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

Розробка системи ранжування авторів за публікаціями та цитуваннями (бекенд, Asp.Net)

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення”

Керівник курсової роботи
ст.в. Сініцина Р. Б.

_____
(підпис)
“____” _____ 2021 р.

Виконав студент Осадчук В. І.
“____” _____ 2021 р.

Київ 2021

Зміст

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мережних технологій,

д.ф.-м.н., професор

_____ Г.І.Малашонок

(підпис)

“____”_____ 20_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Осадчуку Володимиру

4-го курсу факультету інформатики

ТЕМА: Розробка системи ранжування авторів з публікаціями та цитуваннями (бекенд, Asp.Net)

Вихідні дані:

- Веб-Апі, яке виконує ранжування авторів за їх публікаціями та цитуваннями, записує дані у Базу Даних та повертає їх у форматі Json.

Зміст ТЧ до курсової роботи:

1. Вступ
2. Анотація
3. Заповнення даних
4. Індекс Хірша
5. Розширення застосунку
6. Структура проекту
7. Висновки
8. Додатки (за необхідністю)

Дата видачі “___” _____ 202_ р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка системи ранжування авторів з публікаціями та цитуваннями (бекенд, Asp.Net)

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2020 р.	
2.	Огляд літератури за темою роботи	листопад - грудень 2020 р.	
3.	Створення практичної частини роботи	січень - березень 2021 р.	
5.	Написання пояснювальної роботи	січень-березень 2021 р.	
6.	Створення слайдів для доповіді та написання доповіді	лютий-березень 2021 р.	
7.	Надання роботи керівнику для перевірки	березень 2021 р.	
8.	Корегування роботи за результатами перевірки керівником	березень 2021 р.	
9.	Остаточне оформлення пояснювальної роботи та слайдів	березень - квітень 2021 р.	
10.	Захист курсової роботи	?? квітня 2021 р.	

Студент Осадчук В.І. _____

Керівник Сініцина Р.Б. _____

“_____” _____ _____ р.

Вступ

На сьогодні існує багато відкритих загальнодоступних пошукових систем для наукових публікацій та літератури. Наприклад Google Scholar, Microsoft Academic та інші. Завдання таких систем полягає у тому, щоб по певному запиту повертати наукові статті та авторів.

Але, якщо користувачеві потрібно отримати авторів, відсотрованих за певним критерієм, або проранжувати їх, звичайні системи вже не підходять.

Мета даної роботи – створити Базу Даних авторів з їхніми публікаціями та сервіс ранжування авторів, реалізувати їх отримання через API з можливістю сортувати за певним критерієм.

Анотація

Для отримання тестових даних було використано пошукову систему Microsoft Academic, до якої було створено підписку на використання API. Застосунок розроблено на крос-платформному фреймворку для створення веб-застосунків ASP.NET Core. Для збереження даних використано реляційну Базу Даних Ms SQL Server.

Заповнення даних

Джерело даних

Перед тим як ранжувати авторів, потрібно знайти дані, на яких тестувати програму.

Для цього було взято відкриту пошукову систему Microsoft Academic. Після створення підписки на використання їхнього API, було отримано subscription-key для відправки запитів.

Система має чотири публічні методи:

1. CalcHistogram – за певним запитом повертає результат (напр. авторів, статті, журнали) у вигляді гістаграми атрибутів.
2. Evaluate – за певним запитом повертає результат (напр. авторів, статті, журнали) як масив об’єктів.
3. Interpret – інтерпретує запит “людською мовою” на стандартний запит, для двох попередніх методів.
4. Similarity – отримує на вхід дві стрічки та повертає число їх подібності (від -1.00 до 1.00).

Для отримання даних знадобиться лише метод Evaluate.

Також система має сім сутностей, об’єкти яких може повертати:

1. Affiliation
2. Author
3. Conference instance
4. Conference series
5. Field of study
6. Journal
7. Paper

З вищеперелічених знадобляться Author, Journal та Paper.

База даних

Для локального збереження даних використано реляційну Базу Даних Microsoft SQL Server.

Таблиці та поля бази відповідають таким, як у Microsoft Academic. Er-модель наведено нижче:

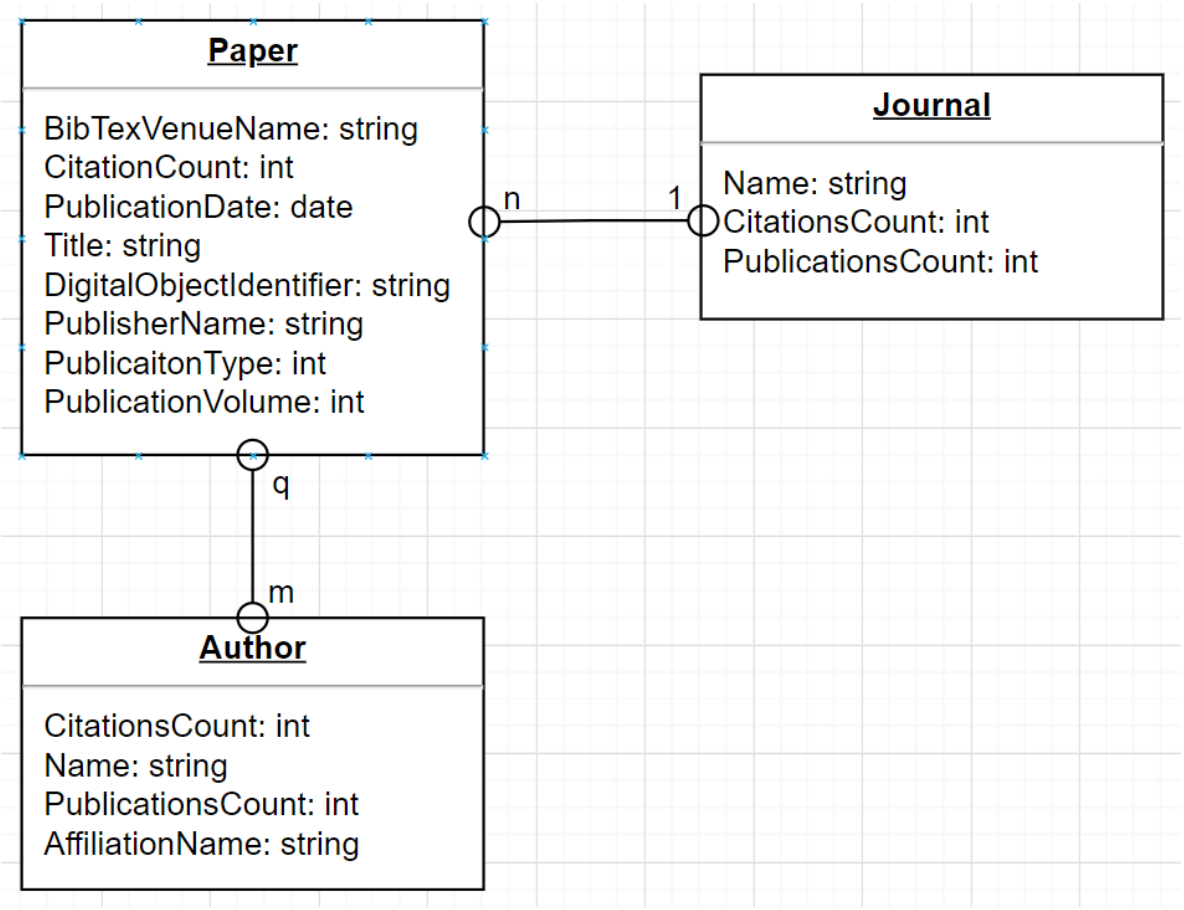


Рисунок 1. ER-модель Бази Даних

Процес заповнення

Для заповнення бази було створено сервіс MicrosoftAcademicService, який відправляє запити на публічний API, десеріалізує результат в клас .Net та повертає результат.

Оскільки застосунок може запускатись з різних пристроїв, та в кожного буде своя База, яка спочатку не має даних, було створено публічний метод, який оброблював POST-запит ‘/api/database/fill’. Він використовує MicrosoftAcademicService для отримання записів, та записує їх в Базу Даних застосунку.

У Microsoft Academic є певні обмеження на кількість запитів до їх API та швидкість отримання результату. А для ранжування одного автора, потрібно мати всі його публікації. Тому процес заповнення Бази не дуже швидкий. База для тестування заповнювалась більше доби та налічує 680 авторів, 38048 журналів та 352535 статей.

Метод працює так, що необов’язково чекати такий чималий проміжок часу для заповнення даних. Якщо його викликати багато разів, він буде перевіряти наявність даних в Базі, та почне з того місця, на якому зупинився при останньому виклику.

Індекс Хірша

Історія

Індекс Хірша (h-індекс) – наукометричний показник, запропонований американським фізиком Хорхе Хірш (університет Сан-Дієго, Каліфорнія), який представляє собою сумарне число посилань на роботи вченого. Критерій заснований на обліку числа публікацій дослідника і числа цитувань цих публікацій.

Індекс було створено для отримання більш адекватної оцінки наукової продуктивності дослідника.

Для прикладу, Хірш вважає, що для дослідника h-index 15-20 відповідає членству в Американському фізичному товаристві, а 45 та вище – в національній академії наук США.

Обчислення

Індекс Хірша – це відношення кількості публікацій дослідника до числа цитувань цих публікацій. Тобто для знаходження індекса певного автора, потрібно знати кількість публікацій та кількість цитувань до кожної з них.

Алгоритм обчислення індесу Хірша наступний: спочатку потрібно відсортувати статті за кількістю їх цитувань в порядку зменшення. Далі знайти статтю, номер якої ідентичний (або менший) до кількості цитат. Цей номер і буде шуканим індексом.

Наприклад, припустимо дослідник має 5 статей з такими кількостями цитувань: 9, 7, 2, 3, 5. Після сортування отримаємо: 9, 7, 5, 3, 2. H-індекс такого автора дорівнює 4, тому, що $9 > 1,\ 7 > 2,\ 5 > 3,\ 3 <= 4$. Якщо ж у автора 100 публікацій, і кожна процитована рівно один раз, його H-індекс дорівнює 1. І якщо в автора одна публікація, що процитована 100 разів, його H-індекс також дорівнює 1.

Реалізація алгоритму

Якщо в базі N авторів, та в кожного з них в середньому M публікацій, то складність обчислення індексу Хірша усіх авторів рівна:

$$N * (M * log_2 M + M)$$

де $M * log_2 M$ – складність сортування.

Тобто загальна складність трохи більша ніж квадратична, а отже, рахувати це при кожному запиті буде дуже затратно, та доведеться багато разів перераховувати одне і теж.

Тому для збереження результату було створено нову таблицю з іменем AuthorRankingResult, та наступними полями:

- AuthorId – ключ на автора
- RankingType – тип ранжування (напр. Хірша)
- Value –результат ранжування

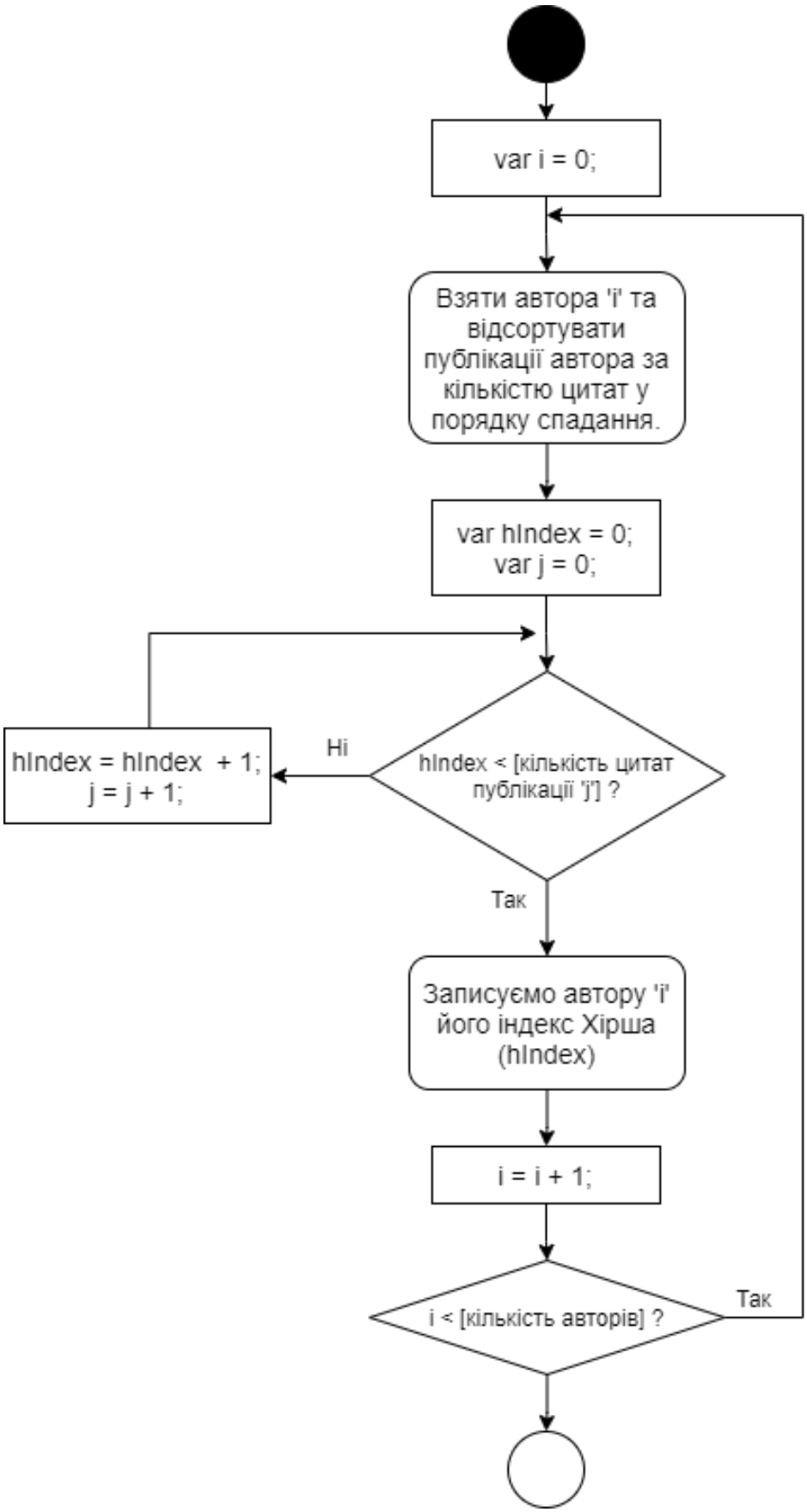
Для виклику ранжування було створено POST-метод ‘api/authors/rank’, що приймає параметр rankingType – тип ранжування (напр. Хірша).

Для ранжування авторів за індексом Хірша було створено сервіс HirshIndex, який наслідує інтерфейс IAuthorsRankingService. Сервіс має наступну реалізацію:

1) Оскільки авторів може бути дуже багато, а разом з автором потрібно мати всі його публікації, створено цикл, в якому оброблюється по 10 авторів (за це відповідає константа AuthorsStep).
2) В циклі робиться запит до таблиці Authors. За допомогою метода Include (метод EF, який транслюється в Join), разом з автором отримуються колекція його публікацій та його AuthorRankingResult.
3) Для кожного автора сортуються його публікації в порядку спадання за полем CitationCount за допомогою метода OrderByDescending.
4) Створюється змінна hIndex зі значенням 0. Цикл проходиться по відсортованим публікаціям, поки hIndex не стане більшою або рівною за кількістю цитат публікації. В циклі hIndex збільшується на 1.

5) Після виходу з циклу, значення, записане в hIndex, і буде індексом Хірша для автора. Воно записується в AuthorRankingResult.
6) Після обробки 10-х авторів зміни зберігаються до БД, та перший цикл починається зпочатку, поки не закінчаться автори.

UML-діаграма алгоритму



Розширення застосунку

На даний момент сервіс ранжує авторів лише за Індексом Хірша, але зазвичай цього недостатньо, і інколи його треба розширити.

Для цього потрібно виконати наступні кроки:

1) Зареєструвати тип

   Для того, щоб застосунок знав про існування нового типу ранжування, потрібно в enum AuthorRankingType додати новий член, вказавши назву типу.

2) Створення сервісу

   Далі потрібно створити сервіс, який буде ранжувати авторів з певною логікою. Його потрібно унаслідувати від інтерфейсу IAuthorsRankingService, та реалізувати його єдиний публічний метод RankAuthors() а також присвоїти тип ранжування через властивість RankingType, який і буде викликатись у разі необхідності.

3) Реєстрація сервісу

   В файлі Startup.cs за допомогою методу AddTransient зареєструвати створений сервіс.

4) Встановлення відповідності

   Для сервісів ранжування було створено делегат

```
public delegate IAuthorsRankingService AuthorsRankingResolver(
    AuthorRankingType rankingType);
```

   який за типом ранжування визначає, який сервіс його виконує.

   Щоб встановити зв’язок між зареєстрованими типом на сервісом, достатньо в файлі Startup.cs, при реєстрації делегату AuthorsRankingResolver додати відповідність в switch:

```
services.AddTransient<AuthorsRankingResolver>(serviceProvider => rankingType =>
{
    return rankingType switch
    {
        AuthorRankingType.HIndex => serviceProvider.GetService<HirschIndex>(),
        _ => throw new KeyNotFoundException()
    };
});
```

Структура проекту

Компоненти

- ClientApp – папка з проектом для проекту Frontend (напр. Angular, React)
- Controllers
  - AuthorsController.cs – містить Api-методи для авторів
  - DatabaseController.cs –містить Api-методи для управління БД
  - PapersController.cs – містить Api-методи для статтей
- Data
  - Migrations – папка з міграціями БД
  - Models
    - Author.cs – модель авторів для БД
    - AuthorRankingResult.cs – модель з результатами ранжування для БД
    - Journal.cs – модель журналів для БД
    - Paper.cs – модель статтей для БД
  - ApplicationContext.cs – сервіс для надсилання запитів до БД. Описує які моделі зберігаються, та їх зв’язки
- Mapping
  - MappingProfile.cs – клас для трансювання однієї моделі в іншу
- Models
  - Dto
    - AuthorsDto.cs – модель автора, яку повертає Microsoft Academic
    - JournalsDto.cs – модель журнала, яку повертає Microsoft Academic
    - PapersDto.cs – модель статті, яку повертає Microsoft Academic

- Enums
    - AuthorRankingType.cs – enum, який містить типи ранжування
    - AuthorSortingType.cs – enum, який вказує, які поля автора можуть сортуватись
    - OrderType.cs – enum, який визначає порядок сортування
- Response
    - AllAuthorsResponse.cs – модель, яку повертає контроллер при отриманні авторів
    - PapersResponse.cs – модель, яку повертає контроллер при отриманні статтей автора

o Services

- AuthorRanking
    - AuthorsRankingResolver.cs – делегат, який встановлює відповідність між типом ранжування та сервісом, який виконує ранжування
    - HirschIndex.cs – сервіс, який ранжує авторів за Індексом Хірша
    - IAuthorsRankingService.cs – інтерфейс, який мають наслідувати сервіси, що виконують ранжування
- AuthorsSortingService.cs – сервіс, який виконує сортування авторів
- DatabaseService.cs – сервіс, для заповнення/очищення Бази Даних
- MicrosoftAcademicService.cs – сервіс, який отримує інформацію з Microsoft Academic

o Program.cs – клас для запуску застосунку

o Startup.cs – клас для реєстрації сервісів та підключення конфігурації

Конфігурація

Задля безпеки, частина конфігурації була додана в User Secrets, тому для запуску застосунку потрібно додати:

- ConnectionStrings
    - SqlConnectionString – стрічка для підключення до Бази Даних Ms Sql Server.
- MsAcademic
    - SubscriptionKey – ключ, отриманий від Microsoft Academic для доступу до API.

Файл конфігурації має формат Json, тому після налаштувань має вийти наступне:

```
{
    "ConnectionStrings": {
        "SqlConnectionString": "[put connection string to sql]"
    },
    "MsAcademic": {
        "SubscriptionKey": "[put subscription key]"
    }
}
```

Публічні API-методи

GET ‘api/authors/all’

Призначення:

Повертає автрорів для певної сторінки (50шт на сторінку), відфільтрованих за іменем та/або організацією, а також відсортованих за певним значенням.

Параметри:

- name – ім’я автора, або частина імені, для фільтрації
- affiliation – організація автора, або частина органіації, для фільтрації
- sortingType – за чим сортувати, значення:
  - 0 – не сортувати
  - 1 – за кількістю публікацій
  - 2 – за кількістю цитат
  - 3 – за обраним ранжуванням (наприклад Хірша)
- orderType – порядок сортування, значення:
  - 0 – за зростанням
  - 1 – за спаданням
- page – сторінка, починаються з 0

Результат:

Повертає масив авторів, поточну сторінку та кількість сторінок.

GET ‘api/authors/{id}’

Призначення:

Повертає автора по унікальному ідентифікатору.

POST ‘api/authors/rank’

Призначення:

Ранжує авторів за певним типом.

Параметри:

- rankingType – Тип ранжування, значення:
  - 1 – Індекс Хірша

Результат:

Час, за який відбувалось ранжування, в секундах.

GET ‘api/papers/byauthor/{authorId}”

Призначення:

Повертає статті певного автора на сторінці, відфільтровані за заголовком, 100шт на сторінці.

Параметри:

- authorId – ІД автора
- title – заголовок, або частина заголовку – для фільтрації
- page – сторінка, починаються з 0

Результат:

Повертає масив статей, поточну сторінку та кількість сторінок.

POST ‘api/database/fill’

Призначення:

Заповнення БД даними з Microsoft Academic. Метод заповнює Базу на 1000 авторів, у кожного з яких буде своя колекція статей та Журналів. Метод виконується досить довго, адже має викликатись лише після підключення до нової БД. Якщо метод перервати, наступного разу заповнення продовжиться з того ж місця.

DELETE ‘api/database/clear’

Призначення:

Повністю очищає Базу Даних.

Висновки

Мету роботи було досягнуто: розроблено Asp.Net застосунок для ранжування авторів. Реалізовано отримання авторів з сервісу Microsoft Academic, запис їх до Бази Даних Ms SQL Server та повернення даних за домогою Web-Api.

Було розглянуто ранжування авторів за Індексом Хірша, його складність та вирішення проблеми з тривалим процесом ранжування багатьох авторів.

Також було проаналізовано рішення з розширення застосунку задля ранжування авторів іншими способами ранжування.

Список джерел

1. https://docs.microsoft.com/en-us/academic-services/project-academic-knowledge/introduction - Документація Microsoft Academic Api.
2. https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1 – документація по Asp.Net Core.
3. https://science.bsu.by/index.php/info/indexes/h-index – індекс Хірша.
4. https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-5.0 – Реалізація запитів в .Net за допомогою HttpClient.
5. https://docs.microsoft.com/en-us/ef/ - Документація по Entity Framework Core.
6. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/working-with-sql?view=aspnetcore-5.0&tabs=visual-studio – Підключення до Ms SQL Server.

Додатки

Ранжування авторів за індексом Хірша.

```
foreach (var author in authors)
{
    var papers :List<Paper> = author.Papers
        .OrderByDescending(p :Paper => p.CitationCount) // IOrderedEnumerable<Paper>
        .ToList();
    var hIndex = 0;
    while (papers.Count > hIndex && hIndex < papers[hIndex].CitationCount)
    {
        hIndex++;
    }

    var authorRankingResult = author.AuthorRankingResults
        .FirstOrDefault(arr :AuthorRankingResult => arr.RankingType == RankingType);

    if (authorRankingResult == null)
    {
        author.AuthorRankingResults.Add( item: new AuthorRankingResult
        {
            RankingType = RankingType,
            Value = hIndex
        });
    }
    else
    {
        authorRankingResult.Value = hIndex;
    }

    _context.Update(author);
}
```

Формування запиту на отримання авторів.

```
// Filter by FullName
var query :IQueryable<Author> = nameArr.Aggregate<string, IQueryable<Author>>(
    _context.Authors.Include( navigationPropertyPath: a :Author => a.AuthorRankingResults),
    (current :IQueryable<Author>, na :string) =>
        current.Where(a :Author => a.FullName.ToLower().Contains(na.ToLower())));

// Filter by Affiliation
query = affiliationArr.Aggregate(query, (current :IQueryable<Author>, af :string) =>
    current.Where(a :Author => a.AffiliationName.ToLower().Contains(af.ToLower())));

// Sort
query = _authorsSortingService.SortAuthors(query, sortingType, orderType);

// Pagination
var authors :List<Author> = await query
    .Skip(AuthorsInPage * page)
    .Take(AuthorsInPage) // IQueryable<Author>
    .ToListAsync(); // Task<List<Author>>
```

Формування відповіді про авторів в залежності від типу ранжування.

```
var authorsResponse = new List<AuthorResponse>();
foreach (var author in authors)
{
    var authorResponse = _mapper.Map<AuthorResponse>(author);
    authorResponse.RankValue = sortingType switch
    {
        AuthorSortingType.HIndex => author.AuthorRankingResults
            .FirstOrDefault(arr :AuthorRankingResult =>
                arr.RankingType == AuthorRankingType.HIndex)
            ?.Value,
        _ => authorResponse.RankValue
    };
    authorsResponse.Add(authorResponse);
}

return new AllAuthorsResponse
{
    Authors = authorsResponse,
    CurrentPage = page,
    TotalPages = authorsCount == 0 ? 0 : (authorsCount - 1) / AuthorsInPage + 1
};
```

Метод на отримання інформації про автора за Id.

```
[HttpGet( template: "{id}")]
Ссылок: 0 | Volodymyr Osadchuk, 40 дн. назад | Автор: 1, изменение: 1
public async Task<Author> GetAuthorById(Guid id)
{
    _logger.Log(LogLevel.Information, message: $"Getting author by id={id}");

    var author = await _context.Authors
        .FirstOrDefaultAsync( predicate: a :Author => a.Id == id);

    return author;
}
```

Заповнення Бази Даних авторами, статтями та журналами.

```
// Create author.
var author = _mapper.Map<Author>(authorDto);

// Get author's papers.
var papersDto = await _msAcademicService.RetrievePapersAsync(
    from: 0, count: 3000, authorId: author.MsId);
await _databaseService.SetAuthorToExistsPapers(papersDto.Papers, author);

// Fill author's papers.
foreach (var paperDto in await _databaseService.NotExistsPapers(papersDto.Papers.ToList()))
{
    var paper = _mapper.Map<Paper>(paperDto);

    // Fill paper's journal
    if (paperDto.Journal?.Id != null)
    {
        var journal = await _databaseService.RetrieveJournal(paperDto.Journal.Id);
        if (journal == null)
        {
            var journalsDto = await _msAcademicService.RetrieveJournalsAsync(
                from: 0, count: 1, msId: paperDto.Journal.Id);
            journal = _mapper.Map<Journal>( source: journalsDto.Journals.First());
        }

        paper.Journal = journal;
        author.Papers.Add(paper);
    }
}

_context.Authors.Add(author);
```

Метод очищення Бази Даних.

```
[HttpDelete( template: "clear")]
Ссылок: 0 | Volodymyr Osadchuk, 45 дн. назад | Автор: 1, изменение: 1
public async Task<ActionResult<int>> ClearAllData()
{
    _logger.Log(LogLevel.Information,  message: "Clearing all data");

    var papers :List<Paper>  = await _context.Papers.ToListAsync();
    _context.RemoveRange(papers);

    var authors :List<Author>  = await _context.Authors.ToListAsync();
    _context.RemoveRange(authors);

    var journals :List<Journal>  = await _context.Journals.ToListAsync();
    _context.RemoveRange(journals);

    await _context.SaveChangesAsync();

    return Accepted();
}
```

Формування запиту на отримання статтей автора.

```
// Filter by title
var query :IQueryable<Paper>  = titleArr.Aggregate<string, IQueryable<Paper>>(
    _context.Papers, (current :IQueryable<Paper> , ti :string ) =>
        current.Where(p :Paper  => p.Title.ToLower().Contains(ti.ToLower())));

// Filter by author
query = query.Where(p :Paper  => p.Authors.Select(a :Author  => a.Id).Contains(authorId));

// Pagination
var papers :List<Paper>  = await query
    .Skip(PapersInPage * page)
    .Take(PapersInPage) // IQueryable<Paper>
    .Include( navigationPropertyPath: p :Paper  => p.Journal) // IIncludableQueryable<Paper,Journal>
    .Include( navigationPropertyPath: p :Paper  => p.Authors) // IIncludableQueryable<Paper,ICollection<
    .ToListAsync(); // Task<List<Paper>>
```

Інтерфейс, що наслідують сервіси ранжування.

```
public interface IAuthorsRankingService
{
    Ссылок: 2 | Volodymyr Osadchuk, 38 дн. назад | Автор: 1, изменение: 1
    public Task RankAuthors();

    Ссылок: 3 | Volodymyr Osadchuk, 38 дн. назад | Автор: 1, изменение: 1
    public AuthorRankingType RankingType { get; }
}
```

Делегат, що повертає сервіс ранжування, в залежності від типу.

```
public delegate IAuthorsRankingService AuthorsRankingResolver(
    AuthorRankingType rankingType);
```

Сортування авторів за полем sortingType та в певному порядку.

```
if (sortingType == AuthorSortingType.None)
{
    return query;
}

Expression<Func<Author, decimal?>> expression = sortingType switch
{
    AuthorSortingType.HIndex => (a :Author =>
        a.AuthorRankingResults.Any(arr :AuthorRankingResult =>
            arr.RankingType == AuthorRankingType.HIndex)
            ? a.AuthorRankingResults.First(arr :AuthorRankingResult =>
                arr.RankingType == AuthorRankingType.HIndex).Value
            : 0),
    AuthorSortingType.PublicationsCount => (a :Author => a.PublicationsCount),
    AuthorSortingType.CitationsCount => (a :Author => a.CitationsCount),
    _ => throw new ArgumentOutOfRangeException(
        nameof(sortingType), sortingType, message: null)
};

query = orderType switch
{
    OrderType.Ascending => query.OrderBy(expression),
    OrderType.Descending => query.OrderByDescending(expression),
    _ => throw new ArgumentOutOfRangeException(nameof(orderType), orderType, message: null)
};

return query;
```

Метод отримання авторів з сервісу Microsoft Academic.

```
public async Task<AuthorsDto> RetrieveAuthorsAsync(
    int from, int count, long msId = -1, string name = null)
{
    var apiUrl:string =
        $"{MsAcademicApiEvaluateUrl}" +
        $"expr=AND(Ty='1'" +
        (msId > 0 ? $", Id={msId}" : "") +
        (!string.IsNullOrEmpty(name) ? $", AuN='{ConvertToNormalizedName(name)}'" : "") +
        $")&" +
        $"offset={from}&" +
        $"count={count}&" +
        $"attributes=Id,CC,DAuN,ECC,PC,LKA.AfId,LKA.AfN&" +
        $"subscription-key={_msAcademicApiSubscriptionKey}";

    var authorsDto = await RetrieveDataAsync<AuthorsDto>(apiUrl);
    return authorsDto;
}
```

Метод отримання статтей з сервісу Microsoft Academic.

```
public async Task<PapersDto> RetrievePapersAsync(
    int from, int count, long authorId = -1, string name = null)
{
    var apiUrl:string =
        $"{MsAcademicApiEvaluateUrl}" +
        $"expr=AND(Ty='0'" +
        (authorId > 0 ? $", Composite(AA.AuId={authorId})" : "") +
        (!string.IsNullOrEmpty(name) ? $", Ti='{ConvertToNormalizedName(name)}'" : "") +
        $")&" +
        $"offset={from}&" +
        $"count={count}&" +
        $"attributes=Id,BV,CC,ECC,D,DN,DOI,PB,Pt,V,AA.AuId,J.JId&" +
        $"subscription-key={_msAcademicApiSubscriptionKey}";

    var papersDto = await RetrieveDataAsync<PapersDto>(apiUrl);
    return papersDto;
}
```

Метод отримання журналів з сервісу Microsoft Academic.

```
public async Task<JournalsDto> RetrieveJournalsAsync(
    int from, int count, long msId = -1, string name = null)
{
    var apiUrl:string =
        $"{MsAcademicApiEvaluateUrl}" +
        $"expr=AND(Ty='2'" +
        (msId > 0 ? $", Id={msId}" : "") +
        (!string.IsNullOrEmpty(name) ? $", JN='{ConvertToNormalizedName(name)}'" : "") +
        $")&" +
        $"offset={from}&" +
        $"count={count}&" +
        $"attributes=Id,CC,DJN,ECC,PC&" +
        $"subscription-key={_msAcademicApiSubscriptionKey}";

    var journalsDto = await RetrieveDataAsync<JournalsDto>(apiUrl);
    return journalsDto;
}
```

Метод відправки запиту на сервіс Microsoft Academic.

```
private async Task<T> RetrieveDataAsync<T>(string apiUrl)
{
    var client = _clientFactory.CreateClient();
    var responseMessage = await client.GetAsync(apiUrl);
    responseMessage.EnsureSuccessStatusCode();

    var responseBody:string = await responseMessage.Content.ReadAsStringAsync();
    return JsonConvert.DeserializeObject<T>(responseBody);
}
```