МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ

Лабораторне заняття №7

Пагінація

Виконав студент групи: МІТ-41

Кухарчук Богдан Петрович

Хід роботи

1. Створення допоміжного класу PaginatedList

Для реалізації пагінації було створено універсальний клас PaginatedList<T>, який успадковується від стандартного List<T>. Цей клас слугує контейнером для даних поточної сторінки та містить метадані навігації (PageIndex, TotalPages).

Ключовим елементом класу є статичний асинхронний метод CreateAsync. Він приймає запит IQueryable, номер сторінки та розмір сторінки. У цьому методі реалізовано ефективну вибірку даних з бази за допомогою формули: source.Skip((pageIndex - 1) * pageSize).Take(pageSize)

Skip: пропускає записи попередніх сторінок.

Take: вибирає лише обмежену кількість записів для поточної сторінки.

Це дозволяє виконувати запит до БД оптимізовано, не завантажуючи всі дані в оперативну пам'ять.

```
using Microsoft.EntityFrameworkCore;

namespace WebApplication1.Models
{

    public class PaginatedList<T> : List<T>
    {
        public int PageIndex { get; private set; } // Номер поточної сторінки
        public int TotalPages { get; private set; } // Загальна кількість сторінок

        public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)
        {
            PageIndex = pageIndex;
            TotalPages = (int)Math.Ceiling(count / (double)pageSize);

            this.AddRange(items); // Додаємо елементи в цей список
        }

        // Властивість: чи є попередня сторінка?
        public bool HasPreviousPage => PageIndex > 1;

        // Властивість: чи є наступна сторінка?
        public bool HasNextPage => PageIndex < TotalPages;

        // Головний метод створення сторінки
        public static async Task<PaginatedList<T>> CreateAsync(IQueryable<T> source, int pageIndex, int pageSize)
        {
            var count = await source.CountAsync(); // Рахуємо загальну кількість записів у БД


            // Skip - пропускаємо (номер сторінки - 1) * розмір сторінки
            // Take - беремо тільки потрібну кількість
            var items = await source.Skip((pageIndex - 1) * pageSize).Take(pageSize).ToListAsync();

            return new PaginatedList<T>(items, count, pageIndex, pageSize);
        }
    }
}
```

Рисунок 7.1. Реалізація класу PaginatedList та методу CreateAsync.

2. Модифікація контролера для підтримки пагінації

У контролері BooksController було змінено метод Index. Тепер він приймає необов'язковий параметр pageNumber. Логіка методу:

Отримується запит IQueryable до таблиці книг через репозиторій (без виконання запиту в БД на цьому етапі).

Викликається метод PaginatedList<Book>.CreateAsync, який виконує SQL-запит із OFFSET та FETCH NEXT і повертає об'єкт пагінованого списку.

Отриманий об'єкт передається у представлення (View).

Розмір сторінки (pageSize) було встановлено рівним 5 для демонстрації роботи пагінації на невеликому наборі даних.

```
// Додаємо параметр pageNumber (за замовчуванням = 1)
public async Task<IActionResult> Index(int? pageNumber)
{
    // Отримуємо запит до БД (але ще не виконуємо його!)

    var booksQuery = _repository.ReadAll<Book>();

    // Визначаємо розмір сторінки
    int pageSize = 5;

    // Використовуємо метод CreateAsync для отримання конкретної сторінки
    // pageNumber ?? 1 означає: якщо pageNumber == null, то беремо 1
    var paginatedBooks = await PaginatedList<Book>.CreateAsync(booksQuery, pageNumber ?? 1, pageSize);

    // Передаємо пагінований список у View
    return View(paginatedBooks);
}
```

Рисунок 7.2. Використання PaginatedList у контролері BooksController.

3. Оновлення представлення (View) та додавання навігації

Було оновлено представлення Index.cshtml. Модель сторінки змінено з IEnumerable<Book> на PaginatedList<Book>. Додано елементи керування навігацією:

Кнопка "Попередня": активна, якщо HasPreviousPage є true.

Кнопка "Наступна": активна, якщо HasNextPage є true.

Індикатор поточної сторінки ("Сторінка X з Y").

Для кнопок використано Tag Helpers (asp-route-pageNumber), які генерують коректні посилання вигляду /Books?pageNumber=2.

Файл: Views/Books/Index.cshtml.

```
@{
    // Логіка для блокування кнопок
    var prevDisabled = !Model.HasPreviousPage ? "disabled" : "";
    var nextDisabled = !Model.HasNextPage ? "disabled" : "";
}

<!-- КНОПКИ ПАГІНАЦІЇ -->
<div class="d-flex justify-content-center gap-2">
    <a asp-action="Index"
       asp-route-pageNumber="@(Model.PageIndex - 1)"
       class="btn btn-outline-primary @prevDisabled">
        &laquo; Попередня
    </a>

    <span class="align-self-center">
        Сторінка @Model.PageIndex з @Model.TotalPages
    </span>

    <a asp-action="Index"
       asp-route-pageNumber="@(Model.PageIndex + 1)"
       class="btn btn-outline-primary @nextDisabled">
        Наступна &raquo;
    </a>
</div>
```

Рисунок 7.3. Реалізація кнопок навігації у представленні Index.cshtml.

Результат роботи

Продемонстровано роботу пагінації на прикладі списку книг.

Перша сторінка: При заході на сторінку списку відображаються перші 5 записів. Кнопка "Попередня" неактивна, кнопка "Наступна" — активна.

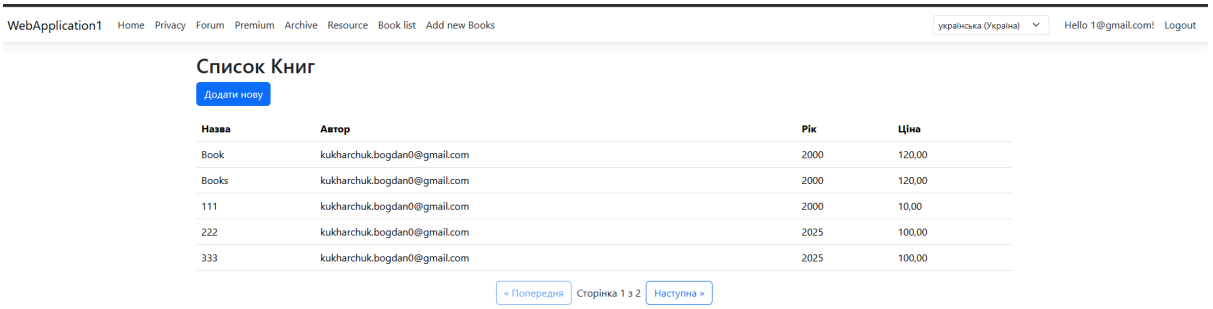

Рисунок 7.4. Відображення першої сторінки списку книг.

Перехід на наступну сторінку: Після натискання кнопки "Наступна" URL змінюється (додається параметр ?pageNumber=2), і відображаються наступні 5 записів. Кнопка "Попередня" стає активною.

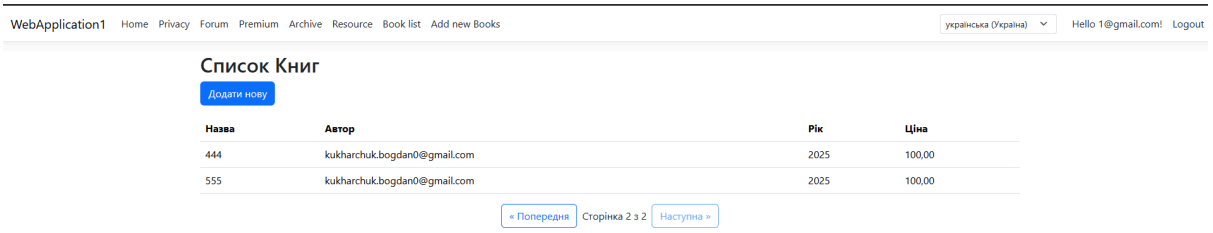

Рисунок 7.5. Відображення другої сторінки з активними кнопками навігації.

Висновок

У ході виконання лабораторної роботи було успішно реалізовано механізм пагінації даних у веб-застосунку ASP.NET Core. Використання методів LINQ Skip та Take дозволило оптимізувати роботу з базою даних, завантажуючи лише необхідну для відображення частину записів, що значно знижує навантаження на сервер та мережу при роботі з великими обсягами інформації. Створений універсальний клас PaginatedList може бути використаний для пагінації будь-яких інших сутностей у проєкті, що забезпечує гнучкість та повторне використання коду. Інтерфейс

користувача було адаптовано для зручної навігації між сторінками даних. Усі зміни зафіксовано в системі контролю версій.