PAGINATION

What is the use of Dapper in .NET core?

Dapper allows you to execute raw SQL queries, map the results to objects, and execute stored procedures, among other things. It is available as a NuGet package. Dapper is lightweight and fast, making it an ideal choice for applications that require low latency and high performance.

What does Dapper do?

Here is a standard ADO.NET C# code for retrieving data from a database and materializing it as a collection of Product objects:

1. var sql = "select \* from products";
2. var products = new List<Product>();
3. using (var connection = new SqlConnection(connString))
4. {
5. connection.Open();
6. using (var command = new SqlCommand(sql, connection))
7. {
8. using (var reader = command.ExecuteReader())
9. {
10. var product = new Product
11. {
12. ProductId = reader.GetInt32(reader.GetOrdinal("ProductId")),
13. ProductName = reader.GetString(reader.GetOrdinal("ProductName")),
14. SupplierId = reader.GetInt32(reader.GetOrdinal("SupplierId")),
15. CategoryId = reader.GetInt32(reader.GetOrdinal("CategoryId")),
16. QuantityPerUnit = reader.GetString(reader.GetOrdinal("QuantityPerUnit")),
17. UnitPrice = reader.GetDecimal(reader.GetOrdinal("UnitPrice")),
18. UnitsInStock = reader.GetInt16(reader.GetOrdinal("UnitsInStock")),
19. UnitsOnOrder = reader.GetInt16(reader.GetOrdinal("UnitsOnOrder")),
20. ReorderLevel = reader.GetInt16(reader.GetOrdinal("ReorderLevel")),
21. Discontinued = reader.GetBoolean(reader.GetOrdinal("Discontinued")),
22. DiscontinuedDate = reader.GetDateTime(reader.GetOrdinal("DiscontinuedDate"))
23. };
24. products.Add(product);
25. }
26. }
27. }

At its most basic level, Dapper replaces the highlighted block of assignment code in the example above with the following:

products = connection.Query<Product>(sql).ToList();

Requirements for pagingation:

Install packages: Dapper

Controller.cs:

[Route("api/[controller]")]

[ApiController]

public class PaginationController : ControllerBase

{

private readonly IPaginationRepo \_paginationRepo;

private readonly IPaginationService \_paginationService;

public PaginationController(

IPaginationRepo paginationRepo,

IPaginationService paginationService

)

{

\_paginationRepo = paginationRepo;

\_paginationService = paginationService;

}

[HttpGet]

[Route("GetStudentsData")]

public async Task<IActionResult> GetStudents( string? SearchTerm, string? SortColumn, string? SortDirection, string? MultiNames, int Page = 1, int Limit = 5)

{

try

{

var result = await \_paginationService.GetStudentsRecords(Page,Limit,SearchTerm,SortColumn,SortDirection,MultiNames);

return Ok(result);

}

catch(Exception ex)

{

return StatusCode(StatusCodes.Status500InternalServerError,new StatusModel

{

StatusCode= 500,

Message = ex.Message,

});

}

}

}

Data Access Layer:

public interface IPaginationRepo

{

Task<StudentResults> GetStudentsRecords(int Page, int Limit,string? SearchTerm,string? SortColumn,string? SortDirection,string? MultiNames);

}

using Dapper;

using DataAccessLayer.Entities;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Data.SqlClient;

using Microsoft.Extensions.Options;

using System;

using System.Buffers;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer.Interfaces

{

public class PaginationRepo : IPaginationRepo

{

private readonly IOptions<ConnectionStrings> \_config;

public PaginationRepo(

IOptions<ConnectionStrings> config

)

{

\_config = config;

}

public async Task<StudentResults> GetStudentsRecords(int Page, int Limit, string? SearchTerm, string? SortColumn, string? SortDirection, string? MultiNames)

{

var conn = \_config.Value.SNCon.ToString();

using IDbConnection connection = new SqlConnection( conn );

// connect multiple querys;

using var multi = await connection.QueryMultipleAsync("pagination\_data", new

{

Page, Limit, SearchTerm, SortColumn, SortDirection, MultiNames

}, commandType: CommandType.StoredProcedure);

// reading the records list to objects student;

var student = await multi.ReadAsync<Student>();

// reading to total records and pages

var paginationData = await multi.ReadFirstAsync<PaginationData>();

return new StudentResults{ Students = student, PaginationData = paginationData };

}

}

}

Business Access Layer:

public interface IPaginationService

{

Task<StudentResults> GetStudentsRecords(int Page, int Limit, string? SearchTerm, string? SortColumn, string? SortDirection, string? MultiNames);

}

public class PaginationService : IPaginationService

{

private readonly IPaginationRepo \_paginationRepo;

public PaginationService(IPaginationRepo paginationRepo) {

\_paginationRepo = paginationRepo;

}

public Task<StudentResults> GetStudentsRecords(int Page, int Limit, string? SearchTerm, string? SortColumn, string? SortDirection, string? MultiNames)

{

var result = \_paginationRepo.GetStudentsRecords(Page, Limit, SearchTerm, SortColumn, SortDirection, MultiNames);

return result;

}

}

--- CREATE PAGINATION PROCS ---

create proc pagination\_data

@Page int =1,

@Limit int =5,

@SearchTerm varchar(100) = null,

@SortColumn varchar(100) = 'Id',

@SortDirection varchar(100)='asc',

@MultiNames varchar(100)=null

as

begin

select \* from Students

where (

@SearchTerm is null or

FirstName like '%'+@SearchTerm+'%' or

LastName like '%'+@SearchTerm+'%'or

DateofBirth like '%' +@SearchTerm+ '%' or

Address like '%'+@SearchTerm+'%' or

RollNumber like '%'+@SearchTerm+'%' or

Marks like '%'+@SearchTerm+'%' or

Grades like '%'+@SearchTerm+'%'

)

and

(@MultiNames is null or [Grades] in (select \* from string\_split(@MultiNames,',')))

order by

case when @SortColumn='Id' and @SortDirection='asc' then Id end,

case when @SortColumn='Id' and @SortDirection='desc' then Id end desc,

case when @SortColumn='FirstName' and @SortDirection='asc' then FirstName end,

case when @SortColumn='FirstName' and @SortDirection='desc' then FirstName end desc,

case when @SortColumn='LastName' and @SortDirection='asc' then LastName end,

case when @SortColumn='LastName' and @SortDirection='desc' then LastName end desc,

case when @SortColumn='DateofBirth' and @SortDirection='asc' then DateofBirth end,

case when @SortColumn='DateofBirth' and @SortDirection='desc' then DateofBirth end desc,

case when @SortColumn='RollNumber' and @SortDirection='asc' then RollNumber end,

case when @SortColumn='RollNumber' and @SortDirection='desc' then RollNumber end desc,

case when @SortColumn='Address' and @SortDirection='asc' then Address end,

case when @SortColumn='Address' and @SortDirection='desc' then Address end desc,

case when @SortColumn='Marks' and @SortDirection='asc' then Marks end,

case when @SortColumn='Marks' and @SortDirection='desc' then Marks end desc,

case when @SortColumn='Grades' and @SortDirection='asc' then Id end,

case when @SortColumn='Grades' and @SortDirection='desc' then Grades end desc

offset ( @Page-1 )\*@Limit ROWS

fetch next @Limit rows only

select count(Id) as TotalRecords,CAST(CEILING((count(Id)\*1.0)/@Limit) as int) as TotalPages

from Students

where (

@SearchTerm is null or

FirstName like '%'+@SearchTerm+'%' or

LastName like '%'+@SearchTerm+'%'or

DateofBirth like '%' +@SearchTerm+ '%' or

Address like '%'+@SearchTerm+'%' or

RollNumber like '%'+@SearchTerm+'%' or

Marks like '%'+@SearchTerm+'%' or

Grades like '%'+@SearchTerm+'%'

)

and

(@MultiNames is null or [Grades] in (select \* from string\_split(@MultiNames,',')))

end

---- pagination limit

exec pagination\_data @Page =2 ,@Limit =4

--- Global search term

exec pagination\_data @SearchTerm ='Saikumar';

--- sorting ascending order

exec pagination\_data @SortColumn='FirstName', @SortDirection='asc'

--- descending order

exec pagination\_data @SortColumn='FirstName', @SortDirection='desc'

EXEC pagination\_data @MultiNames='A,B'