Pagination With Entity Frame Work Core Using Stored Procedures

Create stored procedures:

USE RSA\_GroceryDEV;

SELECT TOP 10 \* from BasketData;

SELECT TOP 10 BasketDataId,Retailer,POSId,OperatorId,TransactionId,TransactionDate,TransactionTotalAmount,TransactionTaxAmount,TransactionTenderType,CreatedDate from BasketData;

--- get proc data from BasketData

alter proc usp\_get\_basketData

as

begin

SELECT TOP 10 BasketDataId,Retailer,POSId,StoreId,OperatorId,TransactionId,TransactionDate,TransactionTime,TransactionTotalAmount,TransactionTaxAmount,TransactionTenderType,CreatedDate from BasketData;

end

exec usp\_get\_basketData;

alter proc usp\_get\_basketData

as

begin

SELECT TOP 10 \* from BasketData;

end

exec usp\_get\_basketData;

alter proc usp\_basket\_pagination

@Page int =1,

@Limit int =10,

@SearchTerm varchar(100) = null,

@SortColumn varchar(100) = 'BasketDataId',

@SortDirection varchar(100)='asc'

as

begin

select \* from BasketData

where(

@SearchTerm is null or

Retailer like '%'+@SearchTerm+'%' or

POSId like '%'+@SearchTerm+'%' or

StoreId like '%'+@SearchTerm+'%' or

OperatorId like '%'+@SearchTerm+'%' or

TransactionId like '%'+@SearchTerm+'%' or

TransactionDate like '%'+@SearchTerm+'%' or

TransactionTime like '%'+@SearchTerm+'%' or

TransactionTotalAmount like '%'+@SearchTerm+'%' or

TransactionTaxAmount like '%'+@SearchTerm+'%' or

TransactionTenderType like '%'+@SearchTerm+'%' or

CreatedDate like '%'+@SearchTerm+'%'

)

order by

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

--- Convert Id int to string format like below

case when @SortColumn='BasketDataId' and @SortDirection='asc' then CONVERT(varchar(255),BasketDataId) end,

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

case when @SortColumn ='BasketDataId' and @SortDirection='desc' then convert(varchar(255),BasketDataId) end desc,

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

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

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

case when @SortColumn ='POSId' and @SortColumn='desc' then POSId end desc,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

fetch next @Limit rows only

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

from BasketData

where(

@SearchTerm is null or

Retailer like '%'+@SearchTerm+'%' or

POSId like '%'+@SearchTerm+'%' or

StoreId like '%'+@SearchTerm+'%' or

OperatorId like '%'+@SearchTerm+'%' or

TransactionId like '%'+@SearchTerm+'%' or

TransactionDate like '%'+@SearchTerm+'%' or

TransactionTime like '%'+@SearchTerm+'%' or

TransactionTotalAmount like '%'+@SearchTerm+'%' or

TransactionTaxAmount like '%'+@SearchTerm+'%' or

TransactionTenderType like '%'+@SearchTerm+'%' or

CreatedDate like '%'+@SearchTerm+'%'

)

end

-- Pagination limit per page 10 records:

exec usp\_basket\_pagination @Page =1 ,@Limit =10

-- GLobal SearchTerm with strings

exec usp\_basket\_pagination @SearchTerm ='SHAWS\_POINTS';

exec usp\_basket\_pagination @SearchTerm ='Cash';

-- Global search for dates like this:

exec usp\_basket\_pagination @SearchTerm ='2018-09-06'

-- Global search for based on Ids:

exec usp\_basket\_pagination @SearchTerm ='365354'

exec usp\_basket\_pagination @SearchTerm ='1096'

--- Global search for datetime:

exec usp\_basket\_pagination @SearchTerm ='2018-08-29'

exec usp\_basket\_pagination @SearchTerm ='16:43:10.0000000'

exec usp\_basket\_pagination @SearchTerm =''

exec usp\_basket\_pagination @SortColumn='Retailer' ,@SortDirection='desc';

exec usp\_basket\_pagination @SortColumn='BasketDataId' ,@SortDirection='desc';

exec usp\_basket\_pagination @SortColumn='BasketDataId' ,@SortDirection='asc';

exec usp\_basket\_pagination @SortColumn='POSId' ,@SortDirection='desc';

exec usp\_basket\_pagination @SortColumn='POSId' ,@SortDirection='asc';

exec usp\_basket\_pagination @SortColumn='TransactionTaxAmount ', @SortDirection='desc' ;

exec usp\_basket\_pagination @SortColumn='TransactionTaxAmount ', @SortDirection='asc' ;

ASP.NET CORE6 WEB API:

public interface IBasketRepository

{

Task<List<BasketData>> GetAllBaskets();

Task<List<BasketData>> GetBasketRecords(int Page, int Limit, string? SearchTerm, string? SortColumn, string? SortDirection);

public ResponseModel GetBasketPaginationData(int Page, int Limit, string? SearchTerm, string? SortColumn, string? SortDirection);

}

ASP.NET CORE6 WEB API CONTROLLER:

Repository:

namespace BasketProject.Repository

{

public class BasketRepository : IBasketRepository

{

private readonly BasketDBContext \_context;

public BasketRepository

(

BasketDBContext context

)

{

\_context = context;

}

public async Task<List<BasketData>> GetAllBaskets()

{

try

{

return await \_context.BasketData.FromSqlRaw<BasketData>("usp\_get\_basketData").ToListAsync();

}

catch(Exception ex)

{

throw new Exception(ex.Message);

}

}

public ResponseModel GetBasketPaginationData(int Page, int Limit, string? SearchTerm, string? SortColumn, string? SortDirection)

{

ResponseModel response = new ResponseModel();

try

{

var parameters = new[]

{

new SqlParameter("@Page", SqlDbType.Int) { Value = Page },

new SqlParameter("@Limit", SqlDbType.Int) { Value = Limit },

new SqlParameter("@SearchTerm", SqlDbType.VarChar, 100) { Value = SearchTerm ?? (object)DBNull.Value },

new SqlParameter("@SortColumn", SqlDbType.VarChar, 100) { Value = SortColumn ?? (object)DBNull.Value },

new SqlParameter("@SortDirection", SqlDbType.VarChar, 100) { Value = SortDirection ?? (object)DBNull.Value }

};

// 10 records per page

var result = \_context.BasketData.FromSqlRaw("EXEC usp\_basket\_pagination @Page, @Limit, @SearchTerm, @SortColumn, @SortDirection", parameters).ToList();

// totalrecords

int totalRecords = \_context.BasketData.Count();

// Calculate total pages

int totalPages = (int)Math.Ceiling((double)totalRecords / Limit);

List<BasketData> basketResuts = new List<BasketData>();

if (result != null)

{

foreach(var item in result)

{

basketResuts.Add(item);

}

response.PaginationData = new PaginationData

{

TotalPages = totalPages,

TotalRecords = totalRecords

};

response.Baskets = basketResuts;

response.StatusMessage = "BasketData has been found";

response.StatusCode = 200;

}

else

{

response.PaginationData = null;

response.StatusMessage = "No BasketData found.";

response.StatusCode = 100;

response.Baskets = null;

}

return response;

}catch ( Exception ex )

{

response.StatusCode = 500;

response.StatusMessage = ex.Message;

throw new Exception(response.StatusMessage);

}

}

public async Task<List<BasketData>> GetBasketRecords(int Page, int Limit, string? SearchTerm, string? SortColumn, string? SortDirection)

{

try

{

var parameters = new[]

{

new SqlParameter("@Page", SqlDbType.Int) { Value = Page },

new SqlParameter("@Limit", SqlDbType.Int) { Value = Limit },

new SqlParameter("@SearchTerm", SqlDbType.VarChar, 100) { Value = SearchTerm ?? (object)DBNull.Value },

new SqlParameter("@SortColumn", SqlDbType.VarChar, 100) { Value = SortColumn ?? (object)DBNull.Value },

new SqlParameter("@SortDirection", SqlDbType.VarChar, 100) { Value = SortDirection ?? (object)DBNull.Value }

};

var result = await \_context.BasketData.FromSqlRaw("EXEC usp\_basket\_pagination @Page, @Limit, @SearchTerm, @SortColumn, @SortDirection", parameters).ToListAsync();

// totalrecords

int totalRecords = \_context.BasketData.Count();

// Calculate total pages

int totalPages = (int)Math.Ceiling((double)totalRecords / Limit);

// Calculate total pages

return result;

}

catch (Exception ex)

{

throw ex;

}

}

}

}

CONTROLLER:

public class BasketController : ControllerBase

{

private readonly IBasketRepository \_basketRepository;

public BasketController(IBasketRepository basketRepository)

{

\_basketRepository = basketRepository;

}

[HttpGet]

[Route("GetBasketPaginationData")]

public IActionResult GetBasketPaginationData(string? SearchTerm, string? SortColumn, string? SortDirection, int Page = 1, int Limit = 10)

{

try

{

//var result = await \_basketRepository.GetBasketRecords(Page, Limit, SearchTerm, SortColumn, SortDirection);

var result = \_basketRepository.GetBasketPaginationData(Page, Limit, SearchTerm,SortColumn, SortDirection);

if(result != null)

{

return Ok(new { data = result.Baskets, TotalRecords = result.PaginationData.TotalRecords, TotalPages = result.PaginationData.TotalPages });

}

return NotFound();

}catch (Exception ex)

{

return StatusCode(StatusCodes.Status500InternalServerError, ex.Message);

}

}

}