# Paginación

```
// GET: api/Productos
[HttpGet]
0 referencias
public async Task<ActionResult<IEnumerable<Producto>>> Get()
{
    //Usuarios
    //Cantidad de registros a devolver por pagina
    //pagina esta solicitado
    //total de pagina
    return await _context.Productos
        .Include(p => p.Usuario).ThenInclude(u => u.Direccion)
        .Where(p => p.Precio > 100)
        .ToListAsync();
}
```

```
private readonly int records =5;

[HttpGet]
0 referencias
public async Task<ActionResult<IEnumerable<Producto>>> Get([FromQuery] int? page)
{
    int _page = page ?? 1;
    int total_records = await _context.Productos.CountAsync();
    int total_pages = Convert.ToInt32(Math.Ceiling(Convert.ToDecimal(total_records /records)));

    var produtos = _context.Productos
        .Skip((_page - 1) * records)
        .Take(records)
        .Include(p => p.Usuario).ThenInclude(u => u.Direccion)
        .Where(p => p.Precio > 100)
        .ToListAsync();

    return Ok(
        new
        {
            pages = total_pages,
            records = produtos,
            current_page = _page,
        });
}
```

# SEARCH

`AsQueryable` es un método que convierte una colección de datos en una **consulta que se puede construir dinámicamente**.

## Explicación sencilla:

- **Sin `AsQueryable`:** Si tienes una lista normal de objetos en memoria (como una lista `List<Producto>`), no puedes agregarle condiciones como filtros (`Where`) y esperar que se comporten como una consulta a la base de datos.
- **Con `AsQueryable`:** Permite que los filtros, ordenamientos o paginaciones se acumulen sin ejecutarse inmediatamente. Esto es útil porque Entity Framework puede traducir todo eso en una **consulta SQL optimizada** para la base de datos.

```
public async Task<ActionResult> SearchProducts(
    [FromQuery] string? name,
    [FromQuery] decimal? minPrice,
    [FromQuery] decimal? maxPrice,
    [FromQuery] int? page)
{
    int _page = page ?? 1;

    // Filtrado dinámico
    var query = _context.Productos.AsQueryable();

    if (!string.IsNullOrEmpty(name))
    {
        query = query.Where(p => p.Nombre.Contains(name));
    }

    if (minPrice.HasValue)
    {
        query = query.Where(p => p.Precio >= minPrice.Value);
    }

    if (maxPrice.HasValue)
    {
        query = query.Where(p => p.Precio <= maxPrice.Value);
    }

    // Total de registros después del filtrado
    int total_records = await query.CountAsync();

    // Calcular total de páginas
    int total_pages = (int)Math.Ceiling(total_records / (decimal)records);

    // Paginación
    var productos = await query
        .Skip((_page - 1) * records)
        .Take(records)
        .Include(p => p.Usuario).ThenInclude(u => u.Direccion)
        .ToListAsync();

    // Construir respuesta
    return Ok(new
    {
        pages = total_pages,
        current_page = _page,
        total_records,
        records = productos
    });
}
```

## PSEUDO CODIGO

```
INICIAR controlador ProductosController

    // Dependencias
    RECIBIR contexto de base de datos (TiendaContext)
    DEFINIR tamaño de página predeterminado (records = 5)

    DEFINIR endpoint HTTP GET en la ruta "/api/productos/search"

        RECIBIR parámetros opcionales desde la URL:
            - name: Nombre del producto a buscar (opcional)
            - minPrice: Precio mínimo (opcional)
            - maxPrice: Precio máximo (opcional)
            - page: Número de página (opcional, predeterminado a 1)

        INICIAR consulta base con todos los productos

        // FILTROS DINÁMICOS
        SI name NO está vacío
            FILTRAR productos donde el nombre contenga el valor de name

        SI minPrice tiene valor
            FILTRAR productos donde el precio sea mayor o igual a minPrice

        SI maxPrice tiene valor
            FILTRAR productos donde el precio sea menor o igual a maxPrice

        // TOTAL DE REGISTROS DESPUÉS DE FILTROS
        CONTAR el número total de registros en la consulta filtrada

        // CÁLCULO DE PÁGINAS
        CALCULAR total_pages como el total de registros dividido por el tamaño de página
(redondeando hacia arriba)

        // PAGINACIÓN
        SALTAR los registros que corresponden a las páginas anteriores
        TOMAR el número de registros correspondiente al tamaño de página
```

INCLUIR información adicional:
- Usuario asociado a cada producto
- Dirección del usuario

OBTENER lista de productos como resultado de la consulta

// RESPUESTA
DEVOLVER un objeto JSON con:
- pages: Total de páginas
- current_page: Página actual
- total_records: Número total de registros filtrados
- records: Lista de productos en la página actual

FIN del endpoint