IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

Filtering dan Searching pada WEB API (ASP.NET CORE)

Junindar, ST, MCPD, MOS, MCT, MVP

junindar@gmail.com

http://junindar.blogspot.com

Abstrak

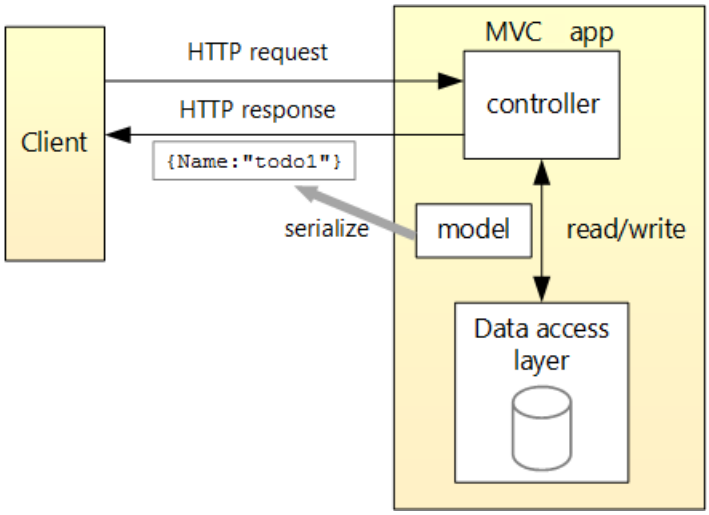
API adalah kepanjangan dari Application Programming Interface yang digunakan perangkat lunak untuk mengakses data, perangkat lunak server atau aplikasi lain dan telah ada selama beberapa waktu.

Sederhananya, API adalah perantara perangkat lunak yang menjembatani dua aplikasi untuk berbicara satu sama lain. Katakanlah API sebagai penerjemah antara dua orang yang tidak berbicara dengan bahasa yang sama, tetapi dapat berkomunikasi menggunakan perantara API.

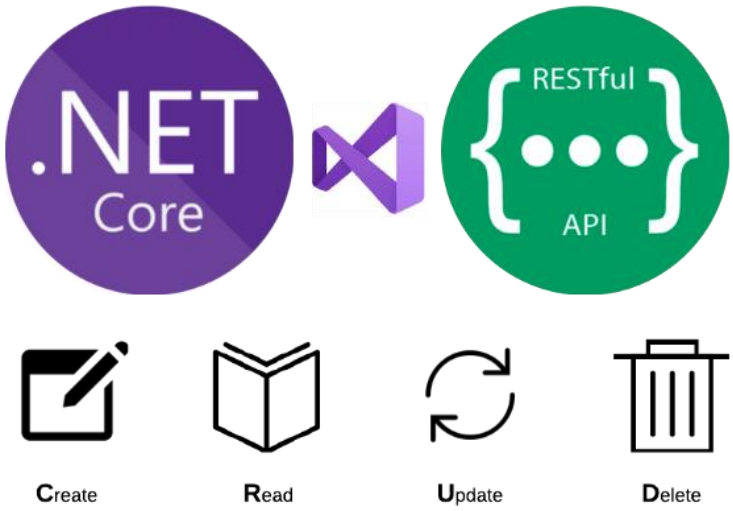
API dapat digunakan pada sistem berbasis web, sistem operasi, sistem basis data, dan perangkat keras komputer.

Pendahuluan

API berkomunikasi melalui serangkaian aturan yang menentukan bagaimana komputer, aplikasi atau mesin dapat berbicara satu sama lain. Web API bertindak sebagai perantara antara dua mesin yang ingin terhubung satu sama lain untuk tugas tertentu.



ASP.NET Core mendukung pembuatan layanan RESTful, juga dikenal sebagai Web API, dengan menggunakan C # sebagai bahasa pemogramannya. Untuk menangani request Web API menggunakan controller. Controller pada Web API adalah class yang berasal ControllerBase.



Pada artikel ini akan dijelaskan Filtering dan Searching pada WEB API. Sebelum kita masuk kedalam materi tersebut, akan dijelaskan terlebih dahulu bagaimana melakukan pengambilan data Parent/Child pada WebAPI. Sebagai contoh pada latihan ini kita memiliki dua buah table, dimana table Categories dan Books memiliki relasi one to many (1 category banyak book). Dan disini akan dijelakan bagaimana melakukan pengambilan data Parent/Child pada WebAPI. Ikuti langkah-langkah dibawah ini.

- Buat sebuah class pada folder Models dengan nama BookDto.

```
public class BookDto
{
    public int Id { get; set; }
    public string Judul { get; set; }
    public string Penulis { get; set; }
    public string Penerbit { get; set; }
    public string Deskripsi { get; set; }
    public bool Status { get; set; }
    public string Gambar { get; set; }
    public int CategoryId { get; set; }
}
```

- Seperti pada Category, kita juga akan menggunakan AutoMapper pada class ini. Tambahkan sebuah class BookProfile pada folder Profiles seperti pada contoh dibawah ini.

```
public class BookProfile : Profile
{
      public BookProfile()
      {
            CreateMap<Book,
            BookDto>().ForMember(dest=>dest.Id,opt=>opt.MapFrom(src=>src.BookID)
            );
     }
}
```

- Dan selanjutnya tambahkan sebuah controller dengan nama BookController. Lalu tambahkan beberapa sintak berikut pada controller

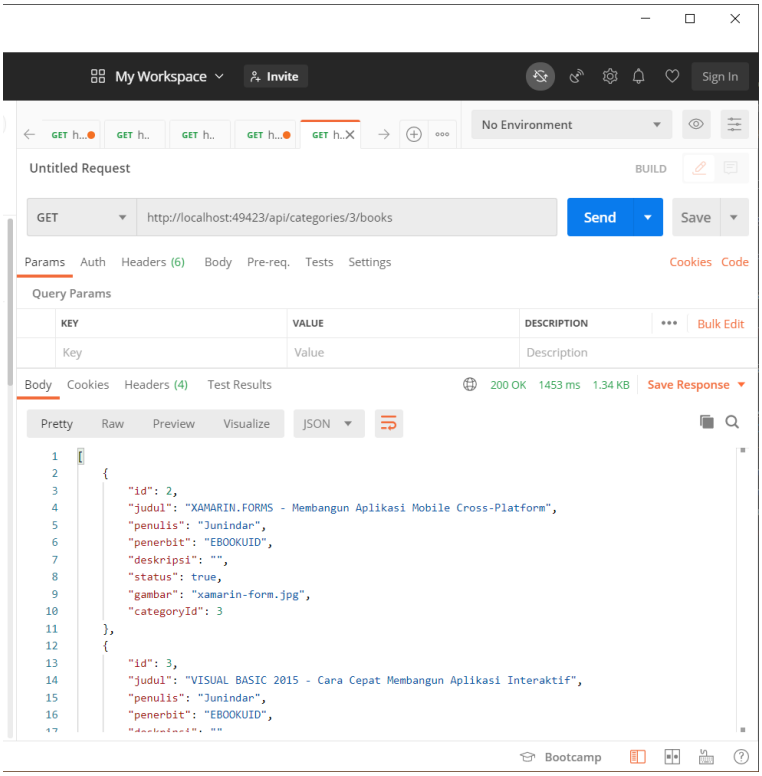
```
private readonly IBookRepository _bookRepository;
private readonly ICategoryRepository _categoryRepository;
private readonly IMapper _mapper;

public BookController(IBookRepository bookRepository, ICategoryRepository
       categoryRepository, IMapper mapper)
{
       _bookRepository = bookRepository;
       _categoryRepository = categoryRepository;
       _mapper = mapper;
}
```

- Disini kita akan mencoba membuat WebAPI untuk menampilkan data buku berdasarkan CategoryId. Tambah sebuah ActionResult dengan nama GetBookByCategory, seperti pada sintak dibawah ini.

```
[HttpGet()]
[Route("api/categories/{categoryId}/books")]
public async Task<ActionResult<IEnumerable<BookDto>>> GetBooksByCategory(
                                                    int categoryId)
{
     var cat = await _categoryRepository.GetById(categoryId);

     if (cat == null)
     {
        return NotFound();
     }
    var resultRepo = await _bookRepository.GetAllBooksByCategoryId(categoryId);
    return Ok(_mapper.Map<IEnumerable<BookDto>>(resultRepo));

}
```

Untuk mendapatkan detail sintaks "GetAllBookByCategoryId" pada class BookRepository dapat dilihat pada project lampiran pada bab Penutup.

- Jalankan program berikut Postman dan pastikan mendapatkan hasil seperti dibawah. Gunakan url berikut http://localhost:49423/api/categories/3/books dan selanjutnya bisa diganti value CategoryId (3) untuk melihat hasil yang lain nya.

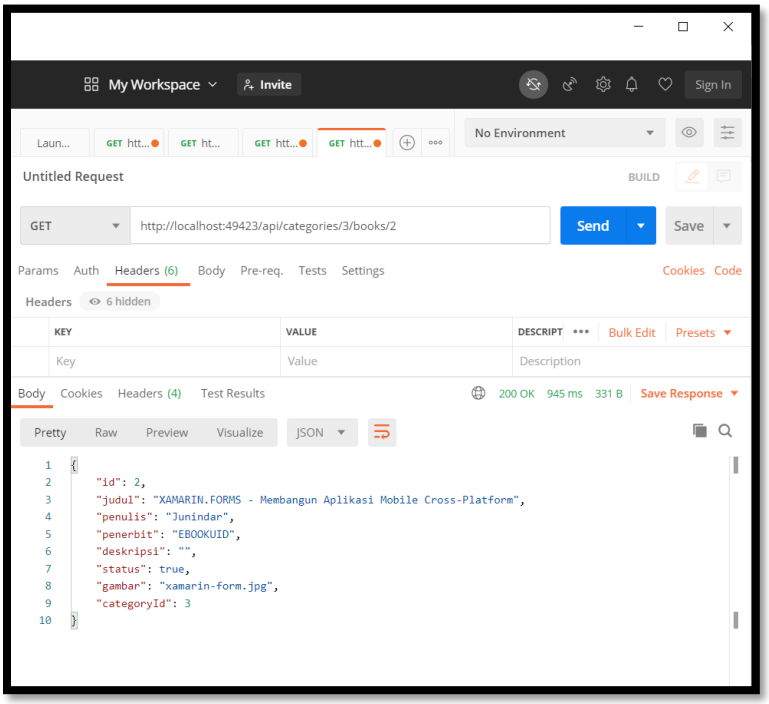


- Setelah berhasil pada latihan diatas, selanjutnya kita akan membuat function untuk mendapatkan single data dari child.
- Tambahkan sebuah ActionResult GetBookByCategory seperti dibawah ini. Pada function ini terdapat dua buah parameter yaitu categoryId dan bookid. Sehingga

akan dilakukan dua kali pengecekan berdasarkan masing-masing paramater.

```
[Route("api/categories/{categoryId}/books/{bookid}")]
public async Task<ActionResult<BookDto>> GetBookByCategory(int categoryId,
                                                           int bookid)
{

    var cat = await _categoryRepository.GetById(categoryId);

    if (cat == null)
    {
       return NotFound();
    }
    var resultRepo = await _bookRepository.GetBookById(bookid);
    if (resultRepo == null)
    {
        return NotFound();
    }

    return Ok(_mapper.Map<BookDto>(resultRepo));
}
```

- Jalankan program berikut Postman dan pastikan mendapatkan hasil seperti dibawah. Gunakan url berikut http://localhost:49423/api/categories/3/books/2 dan selanjutnya bisa diganti value dari bookid (2) untuk melihat hasil yang lain nya.



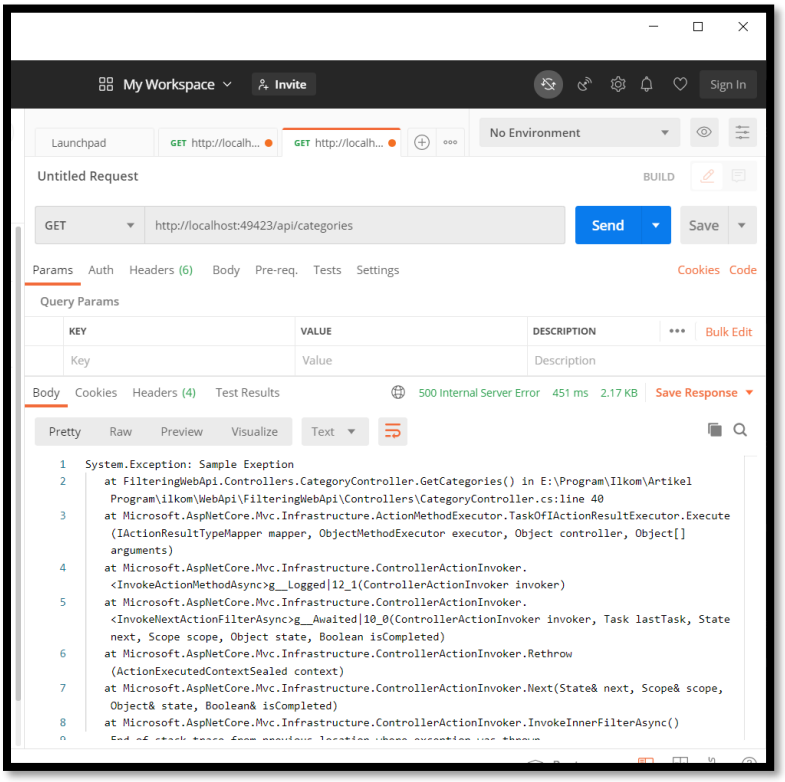
Handling Error

Pada sebuah program penanganan kesalahan sangat penting sekali. Pada saat development kita akan mengetahui dimana proses yang mengalami masalah. Sehingga proses troubleshooting bisa dilakukan dengan lebih cepat.

Buka CategoryController dan ganti sintaksnya seperti dibawah ini (tambahkan sintaks throw new Exception)

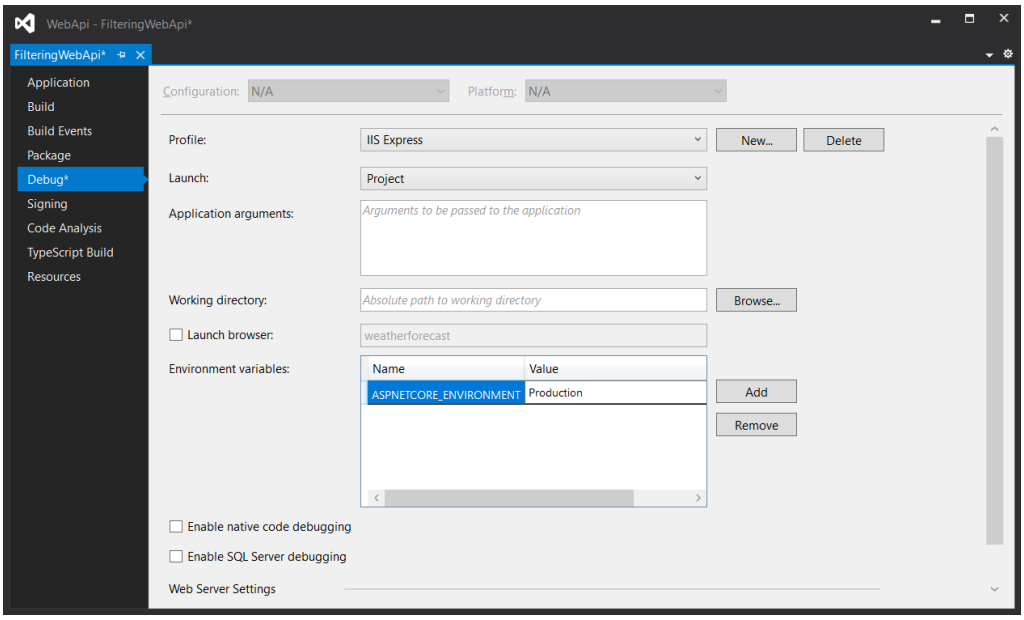
```
[HttpGet()]
public async Task<IActionResult> GetCategories()
{
            throw new Exception("Sample Exeption");
            var resultRepo = await _categoryRepository.GetAll();
            return Ok(_mapper.Map<IEnumerable<CategoryDto>>(resultRepo));


}
```

Selanjutnya coba jalankan dan requets categories pada Web API (http://localhost:49423/api/categories)



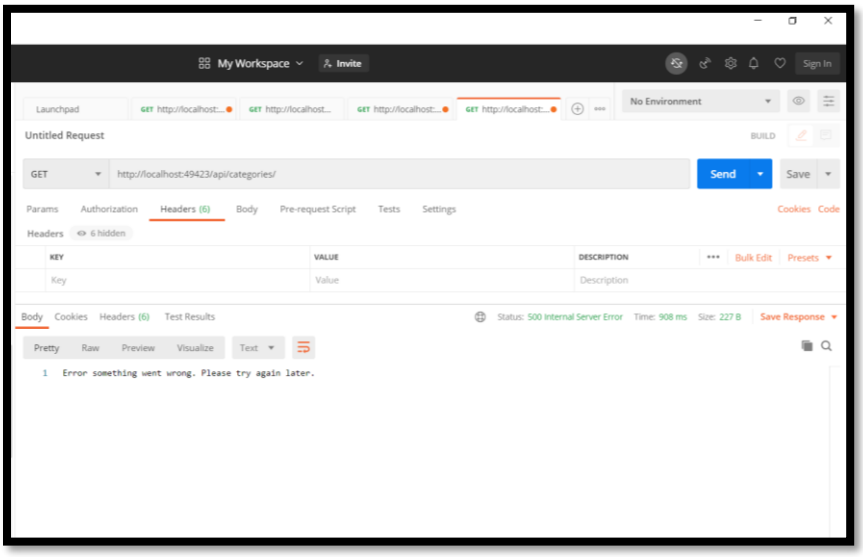
Dari gambar diatas dapat kita lihat pesan error yang dikirimkan oleh Web API. Tapi bagaimana jika Web API sudah ada di production. Tentunya error seperti ini tidak disarankan. Untuk mengatasi hal tersebut kita perlu melakukan beberapa langkah, seperti berikut.

- Buka jendela Project Properties lalu klik tab Debug. Dan pada Environment variables ganti value menjadi “Production“.



- Selanjutnya file Startup.cs dan tambahkan kondisi seperti dibawah ini. Sehingga jika pada production error yang akan tampil adalah .“Error something went wrong“

```
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
     app.UseExceptionHandler(appBuilder =>
     {
         appBuilder.Run(async context =>
         {
              context.Response.StatusCode = 500;
               await context.Response.WriteAsync("Error something went wrong.
               Please try again later.");
           });
        });
}
```

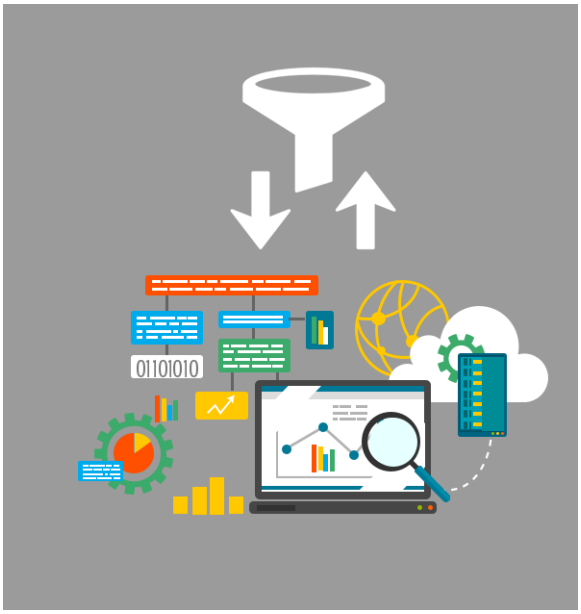


Setelah berhasil ganti kembali value menjadi Development.

Filter vs Search

Filter adalah proses pengambilan data dengan menyeleksi atau menyaring berdasarkan suatu criteria tertentu. . Sebagai contoh kita ingin mendapatkan seluruh data buku dengan Penerbit-nya adalah “Ebookuid” atau “Skripta”. Maka kita akan mengirimkan value (“Ebookuid” atau “Skripta”) untuk field Penerbit dan mengambil data yang sama dengan value.

Search adalah proses pencarian data dengan menggunakan keyword. Search cakupan-nya jauh lebih luas. Sebagai contoh kita mencari kata “Visual” pada table Books.



Untuk lebih memudahkan dalam memhami isi artikel kita akan mencoba langsung membuat dua fungsi tersebut (Filtering dan Searching) ikuti langkah-langkah dibawah ini.

- Tambahkan sintaks dibawah pada file IBookRepository.

```
Task<IEnumerable<Book>> GetAllBooks();
Task<IEnumerable<Book>> GetAllBooks(string penerbit);
```

- Sedangkan untuk implementasi kedua function diatas dilakukan pada class BookRepository, seperti dibawah.

```
public async Task<IEnumerable<Book>> GetAllBooks()
{

     return await _dbContext.Books.Include(b => b.Category).ToListAsync();
}

public async Task<IEnumerable<Book>> GetAllBooks(string penerbit)
{
      if (string.IsNullOrEmpty(penerbit))
      {
          return await GetAllBooks();
      }

      return await _dbContext.Books.Where(c=>c.Penerbit==penerbit).Include(b =>
      b.Category).ToListAsync();
}
```

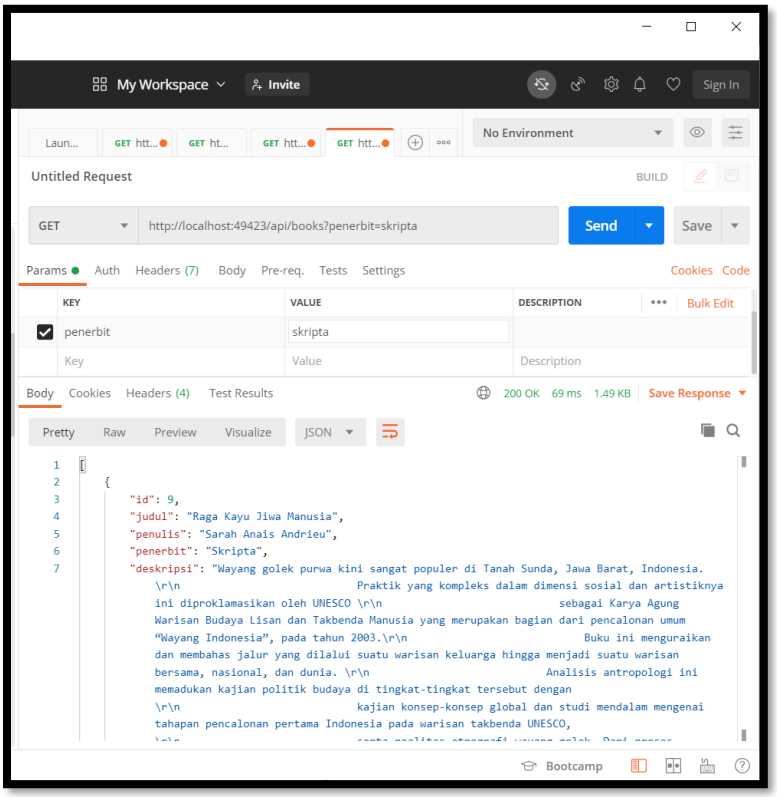
Function pertama digunakan untuk menampilkan seluruh data Books sedangkan yang kedua terdapat sebuah paramater penerbit, dimana value nya akan digunakan untuk proses filter data pada field Penerbit. Sebelum dilakukan proses filtering paramater tersebut akan dicek terlebih dahulu valuenya. Jika paramater tersebut

sama dengan kosong maka akan langsung memanggil function GetAllBooks, yaitu function pertama (menampilkan seluruh data Books).

- Pada BookController tambahkan sebuah ActionResult seperti dibawah ini.

```
[HttpGet()]
public async Task<ActionResult<IEnumerable<BookDto>>> GetBooks(string penerbit)
{

    var resultRepo = await _bookRepository.GetAllBooks(penerbit);
    return Ok(_mapper.Map<IEnumerable<BookDto>>(resultRepo));
}
```

Selanjutnya coba jalankan dan requets books pada Web API (http://localhost:49423/api/books) lalu pada Postman Tab Params untuk KEY ketikkan **penerbit**, sedangkan valuenya **skripta.** Pastikan hasilnya daftar buku dengan penerbitnya adalah skripta.



- Setelah berhasil akan kita lanjutkan dengan fungsi search. Tambahkan sintaks dibawah pada file IBookRepository.

```
Task<IEnumerable<Book>> GetAllBooks(string penerbit,string keyword);
```

- Sedangkan untuk implementasi function diatas dilakukan pada class BookRepository, seperti dibawah.

```
public async Task<IEnumerable<Book>> GetAllBooks(string penerbit,string keyword)
{
      if (string.IsNullOrEmpty(penerbit) && string.IsNullOrEmpty(keyword))
      {
          return await GetAllBooks();
      }

      var books = _dbContext.Books.AsQueryable();

      if (!string.IsNullOrEmpty(penerbit))
      {
          books = books.Where(c => c.Penerbit==penerbit);
      }

      if (!string.IsNullOrEmpty(keyword))
      {
                books = books.Where(c => c.Judul.Contains(keyword) ||
                c.Penulis.Contains(keyword) ||
                c.Penerbit.Contains(keyword) || c.Deskripsi.Contains(keyword));
      }

     return books.ToList();

}
```

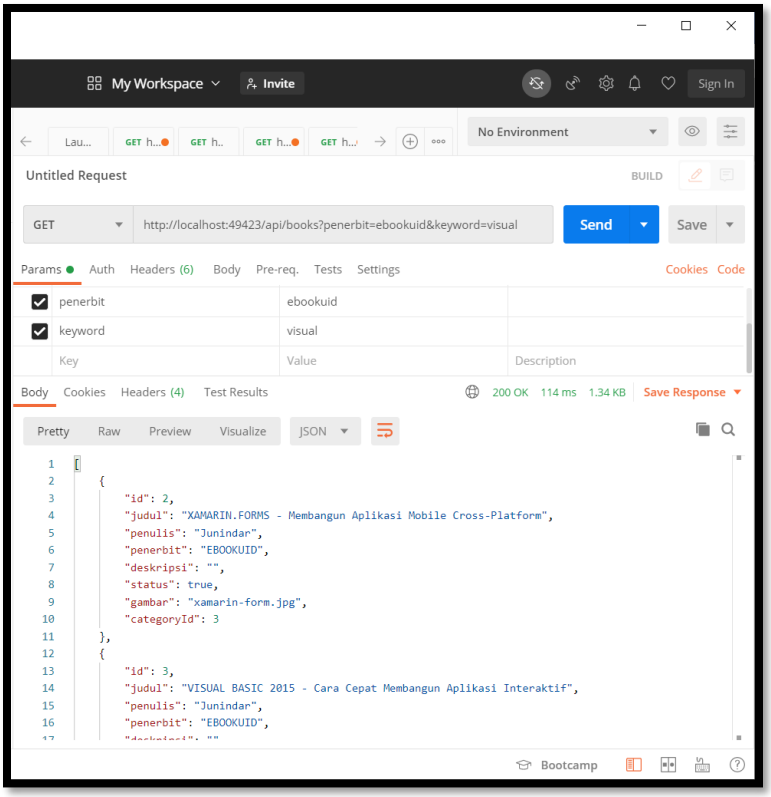
Sebelum kita melakukan proses searching maupun filtering sebelumnya kita cek terlebih dahulu value dari kedua parameter. Jika kedua parameter sama dengan kosong maka akan langsung memanggil function GetAllBooks, yaitu function pertama (menampilkan seluruh data Books). Disini kita akan menggunakan IQueryable. IQueryable menggunakan name space System.Linq dan akan mengeksekusi “select query“ pada server dengan seluruh filter yang telah kita tentukan sebelumnya. Bisa dilihat pada sintaks diatas, terdapat beberapa kondisi untuk membuat proses filtering dan Searching.

Selanjutnya ganti ActionResult GetBooks seperti dibawah.

```
[HttpGet()]
[Route("api/books")]
public async Task<ActionResult<IEnumerable<BookDto>>> GetBooks(string penerbit,
                                                               string keyword)
{

     var resultRepo = await _bookRepository.GetAllBooks(penerbit,keyword);
     return Ok(_mapper.Map<IEnumerable<BookDto>>(resultRepo));
}
```

Perbedaannya adalah pada function ini terdapat dua buah parameter yaitu penerbit dan keyword dan juga memanggil function GetAllBooks yang menggunakan dua paramater yang baru saja kita buat diatas.

Selanjutnya coba jalankan dan requets books pada Web API (http://localhost:49423/api/books) lalu pada Postman Tab Params untuk KEY ketikkan **penerbit** dan **keyword** sedangkan valuenya **ebookuid** dan **visual.** Pastikan hasilnya daftar buku dengan penerbitnya adalah **ebookuid** dan yang mengandung kata **visual.**



Grouping Action Paramaters

Pada action result diatas kita telah membuat dengan menggunakan dua buah paramater. Bagaimana jika kedepannya parameternya bertambah, maka function pada Repository pun akan berubah juga paramaternya. Dengan meletakkan parameter kedalam sebuah object maka tidak ada ketergantungan parameter antara action dan function pada repository.

- Tambahkan sebuah folder Paramaters pada project, lalu tambahkan sebuah class dengan nama BooksParameters seperti dibawah.

```
public class BooksParameters
{
     public string Penerbit { get; set; }
     public string Keyword { get; set; }
}
```

- Tambahkan sintaks dibawah pada file IBookRepository.

```
Task<IEnumerable<Book>> GetAllBooks(BooksParameters booksParameters);
```

- Sedangkan untuk implementasi function diatas dilakukan pada class BookRepository, seperti dibawah.

```
public async Task<IEnumerable<Book>> GetAllBooks(BooksParameters booksParameters)
{
     if (string.IsNullOrEmpty(booksParameters.Penerbit) &&
         string.IsNullOrEmpty(booksParameters.Keyword))
     {
         return await GetAllBooks();
     }

     var books = _dbContext.Books.AsQueryable();

     if (!string.IsNullOrEmpty(booksParameters.Penerbit))
     {
        books = books.Where(c => c.Penerbit == booksParameters.Penerbit);
     }

    if (!string.IsNullOrEmpty(booksParameters.Keyword))
    {
         books = books.Where(c => c.Judul.Contains(booksParameters.Keyword)
                              || c.Penulis.Contains(booksParameters.Keyword) ||
                                        c.Penerbit.Contains(booksParameters.Keyword) ||
                                        c.Deskripsi.Contains(booksParameters.Keyword));
    }

   return books.ToList();
}
```

Sebenarnya sintaksnya sama seperti pada function sebelumnya, hanya kita ganti paramaternya saja.

Selanjutnya ganti ActionResult GetBooks seperti dibawah.

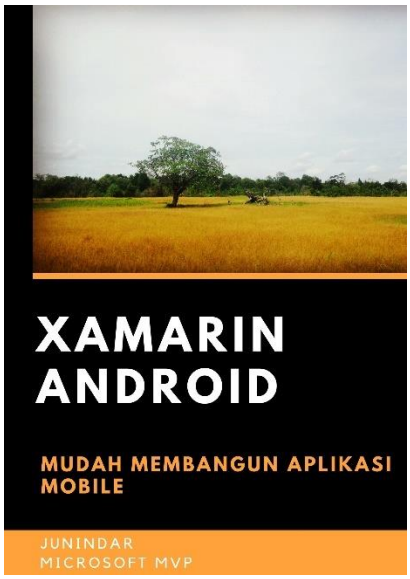
```
[HttpGet()]
[Route("api/books")]
public async Task<ActionResult<IEnumerable<BookDto>>> GetBooks(
         [FromQuery] BooksParameters booksParameters)
{

     var resultRepo = await _bookRepository.GetAllBooks(booksParameters);
     return Ok(_mapper.Map<IEnumerable<BookDto>>(resultRepo));
}
```

Pastikan hasilnya sama seperti pada action result sebelumnya.

Penutup

Sedangkan untuk memudahkan dalam memahami isi artikel, maka penulis juga menyertakan dengan full source code project latihan ini, dan dapat di download disini http://junindar.blogspot.com/2020/10/filtering-dan-searching-pada-web-api.html

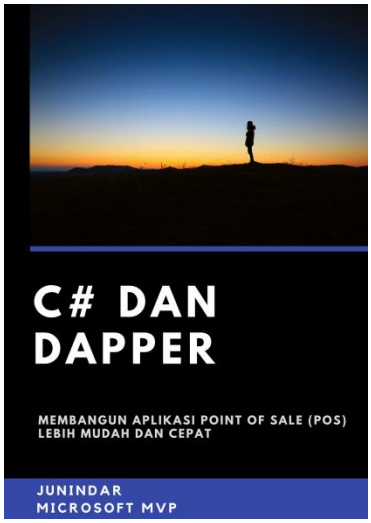
Referensi



https://play.google.com/store/books/details?id=G4tFDgAAQBAJ

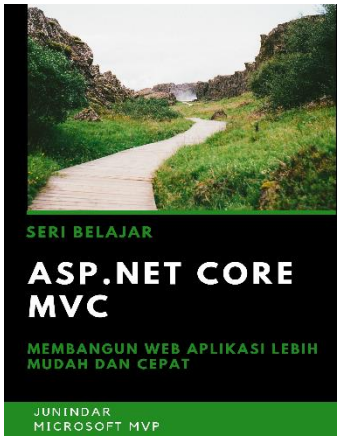
https://play.google.com/store/books/details?id=VSLiDQAAQBAJ

https://play.google.com/store/books/details/Junindar_Xamarin_Forms?id=6Wg-DwAAQBAJ



https://play.google.com/store/books/details/Junindar_C_dan_Dapper_Membangun_Aplikasi_POS_Point?id=6TErDwAAQBAJ

https://play.google.com/store/books/details/Junindar_ASP_NET_MVC_Membangun_Aplikasi_Web_Lebih?id=XLlyDwAAQBAJ



https://play.google.com/store/books/details/Junindar_ASP_NET_CORE_MVC?id=xEe5DwAAQBAJ

Biografi Penulis.

Junindar Lahir di Tanjung Pinang, 21 Juni 1982. Menyelesaikan Program S1 pada jurusan Teknik Inscreenatika di Sekolah Tinggi Sains dan Teknologi Indonesia (ST-INTEN-Bandung). Junindar mendapatkan Award Microsoft MVP VB pertanggal 1 oktober 2009 hingga saat ini. Senang mengutak-atik computer yang berkaitan dengan bahasa pemrograman. Keahlian, sedikit mengerti beberapa bahasa pemrograman seperti : VB.Net, C#, SharePoint, ASP.NET, VBA. Reporting: Crystal Report dan Report Builder. Database: MS Access, MY SQL dan SQL Server. Simulation / Modeling Packages: Visio Enterprise, Rational Rose dan Power Designer. Dan senang bermain gitar, karena untuk bisa menjadi pemain gitar dan seorang programmer sama-sama membutuhkan seni. Pada saat ini bekerja di salah satu Perusahaan Consulting dan Project Management di Malaysia sebagai Senior Consultant. Memiliki beberapa sertifikasi dari Microsoft yaitu Microsoft Certified Professional Developer (MCPD – SharePoint 2010), MOS (Microsoft Office Specialist) dan MCT (Microsoft Certified Trainer) Mempunyai moto hidup: "**Jauh lebih baik menjadi Orang Bodoh yang giat belajar, dari pada orang Pintar yang tidak pernah mengimplementasikan ilmunya".**