IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

Membuat proses CRUD pada WEB API (ASP.NET CORE)

Junindar, ST, MCPD, MOS, MCT, MVP

junindar@gmail.com
http://junindar.blogspot.com

Abstrak

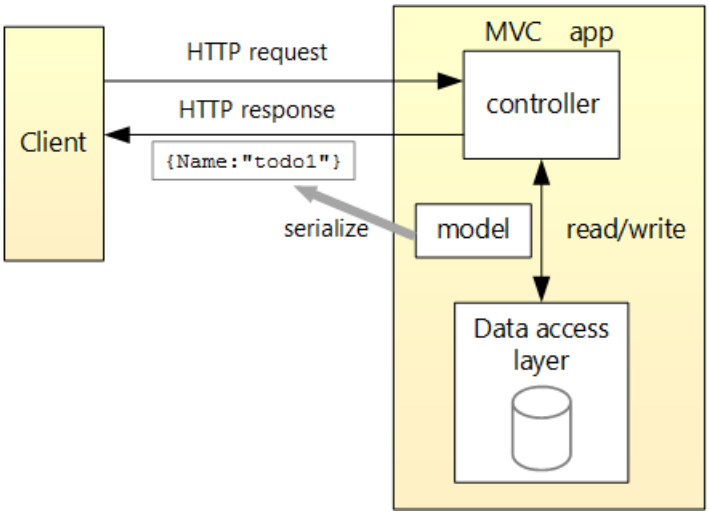
API adalah kepanjangan dari Application Programming Interface yang digunakan perangkat lunak untuk mengakses data, perangkat lunak server atau aplikasi lain dan telah ada selama beberapa waktu.

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.

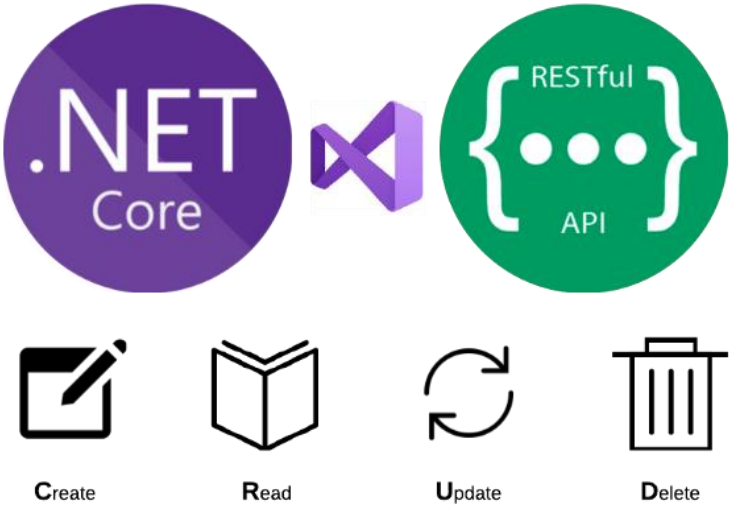
API dapat digunakan pada sistem berbasis web, sistem operasi, sistem basis data, dan perangkat keras komputer.

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.



Seperti pada aplikasi pada umumnya, pada webapi kita dapat juga membuat proses CRUD. Pada latihan sebelumnya telah dijelaskan bagaimana melakukan filtering dan searching pada webapi. Seperti pada latihan-latihan sebelumnya artikel ini akan mengajak pembacanya untuk mempraktekan langsung kedalam project pada Visual Studio. Pastikan sebelumnya sudah membaca artikel-artikel terdahulu, agar lebih mengerti konsep webapi pada artikel ini.

- POST : HTTP POST digunakan untuk melakukan proses penambahan record baru pada sumber data. Ikuti langkah-langkah dibawah ini untuk membuat action result POST.

```
[HttpPost]
public async Task<ActionResult<CategoryDto>> CreateCategory(CategoryDto category)
{
      var categoryEntity = _mapper.Map<Category>(category);
      categoryEntity = await _categoryRepository.InsertCategory(categoryEntity);

      var categoryForReturn = _mapper.Map<CategoryDto>(categoryEntity);

     return CreatedAtRoute("GetCategory", new { categoryId = categoryForReturn.Id },
     categoryForReturn);
}
```

pada sintaks diatas kita tambah sebuah action result dengan nama CreateCategory dengan parameternya CategoryDto, dimana telah kita buat pada artikel sebelumnya. Pada baris pertama didalam method tersebut, kita melakukan mapping object (CategoryDto) menjadi object Category. Hasil dari mapping tersebut akan dikirim ke method InsertCategory. Berikut sintak InsertCategory yang digunakan.

```
public async Task<Category> InsertCategory(Category obj)
{
        _dbContext.Add(obj);
        await _dbContext.SaveChangesAsync();
        return obj ;
}
```

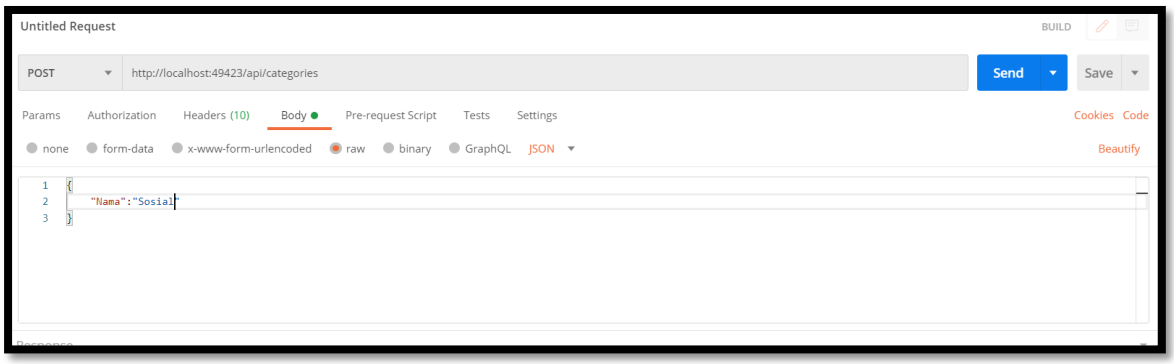
InsertCategory adalah sebuah function dimana method ini memiliki nilai kembalian dari hasil proses insert kedalam table. Lalu hasil dari proses InsertCategory ini akan dilakukan mapping kembali. Jika mapping pertama adalah dari CategoryDto ke Category, sedangkan ini kebalikannya. Dan pastikan tambahkan sintaks dibawah ini pada CategoryProfile.

```
CreateMap<CategoryDto, Category>().ForMember(dest => dest.CategoryID, opt =>
                opt.MapFrom(src => src.Id))
                .ForMember(dest => dest.NamaCategory, opt => opt.MapFrom(src => src.Nama)
);
```

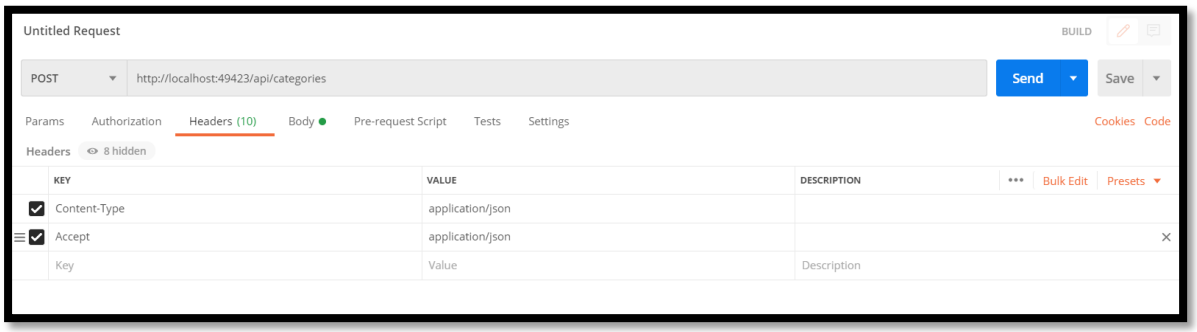
Dan terkahir adalah sintaks return pada action result ini. Disini kita akan memanggil action result lain yaitu GetCategory dengan mengirimkan data hasil dari mapping diatas. Pastikan pada GetCategory ditambahkan attribute name seperti dibawah ini.

```
[HttpGet("{categoryId}",Name = "GetCategory")]
```

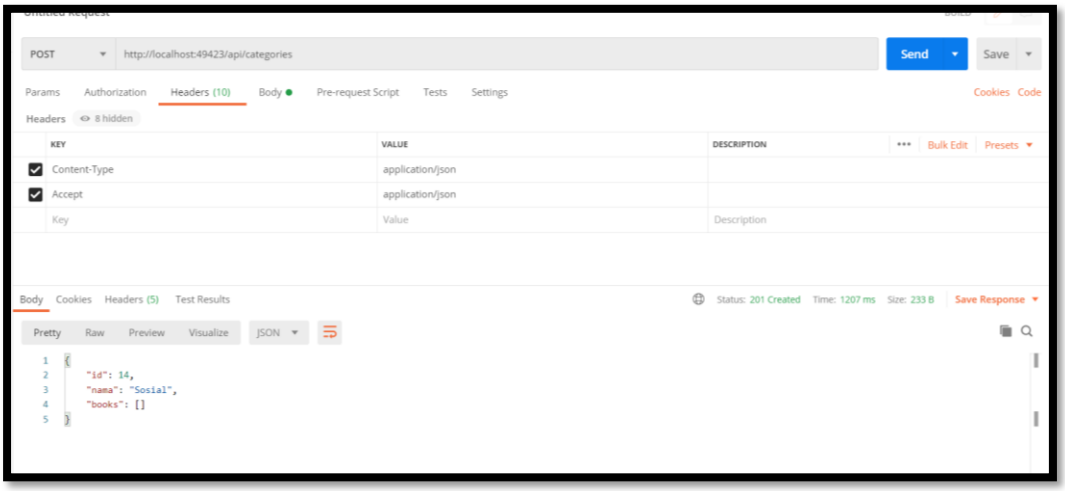
Jalankan program dan lakukan proses insert dengan menggunakan postman seperti dibawah.



Pada Postman klik tab Body lalu pilih radio button “raw“. Dan ketikkan data dengan format json seperti diatas.



Untuk tab Header, tambahkan dua buah KEY : Content-Type dan Accept dengan valuenya adalah “application/json“. Dan pastikan method yang dipilih ada POST, selanjutnya tekan button Send. Pastikan data diatas masuk kedalam table, dan mendapatkan hasil seperti dibawah ini.



Setelah berhasil dengan latihan diatas, maka akan kita lanjutkan lagi dengan membuat proses insert pada child. Seperti kita ketahui bahwa antara table Categories dan Books memiliki relasi parent-child. Dimana satu category dapat memiliki lebih dari satu Book (ont to many). Ikuti langkah-langkah dibawah ini.

- Tambakan sebuah action dengan nama “CreateBookForCategory“ dan tambahkan atribut route (api/categories/{categoryId}/books) pada action tersebut. Untuk sintaks detailnya seperti dibawah ini.

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

      if (cat == null)
      {
          return NotFound();
      }
     var bookEntity = _mapper.Map<Book>(book);
     bookEntity.CategoryID = categoryId;
     bookEntity = await _bookRepository.Insert(bookEntity);

     var bookForReturn = _mapper.Map<BookDto>(bookEntity);

     return CreatedAtRoute("GetBookByCategory", new { categoryId = categoryId,
                bookid=bookForReturn.Id }, bookForReturn);
}
```

Pada action ini memiliki dua buah parameter yaitu “categoryId“ bertipe integer dan “book“ dari object class BookDto. Pada baris awal, kita panggil method GetById dari class categoryRepository, yang digunakan untuk mencari data category berdasarkan categoryid. Jika hasil dari pencarian tidak ditemukan maka

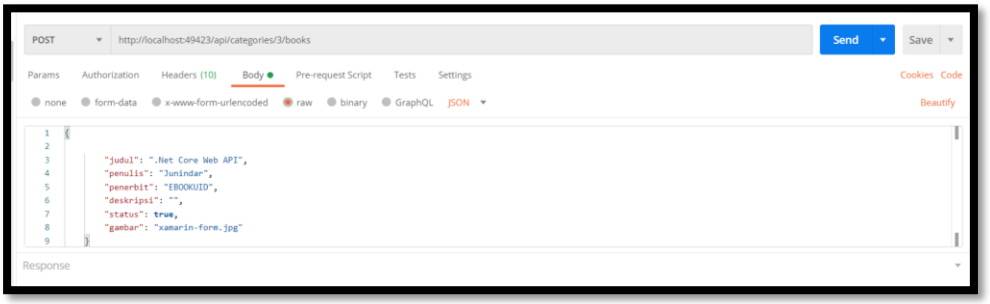
return dari action adalah “NotFound“. Jika data yang dicari ditemukan, maka langkah selanjutnya adalah melakukan mapping dari object class BookDto menjadi Book. Dilanjutkan dengan memasukkan data CategoryID dari hasil mapping sebelumnya dengan nilai categoryId pada parameter action ini.

Selanjutnya kita panggil method Insert dari class bookRepository. Untuk melihat detail method Insert dapat dilihat di project lampiran. Yang perlu diperhatikan method ini berupa function yang nilai kembaliannya adalah hasil dari data proses insert itu sendiri. Sehingga nantinya hasil dari proses insert itu akan dilakukan mapping menjadi ke object class BookDto kembali. Pastikan sebelumnya untuk mengetikkan sintaks dibawah pada class BookProfile.

```
CreateMap<BookDto, Book>().ForMember(dest => dest.BookID, opt =>
opt.MapFrom(src => src.Id));
```

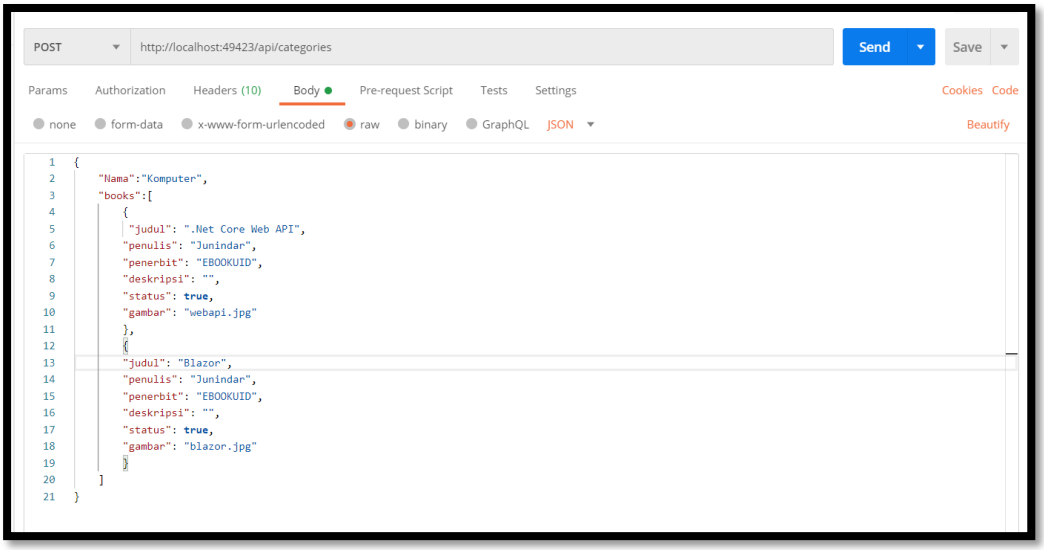
Jika sudah selesai dengan langkah-langkah diatas, sekarang saatnya kita untuk melakukan pengetesan.

/api/categories/3/books

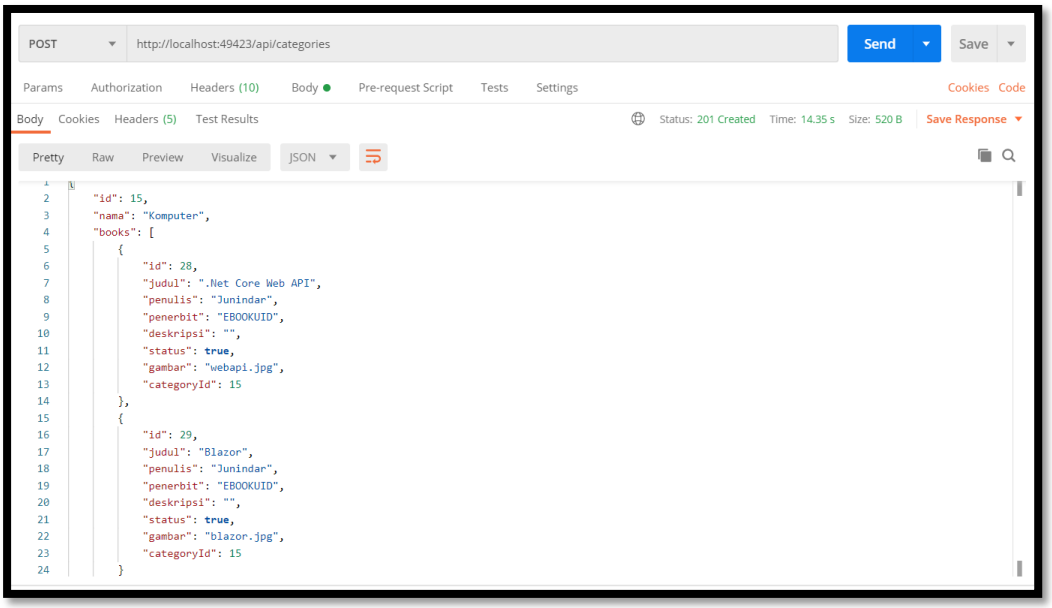


Lalu bagaimana jika kita akan melakukan proses Insert Parent Child dalam satu request? Untuk melakukan proses ini hal yang perlu kita lakukan adalah dengan menambah sebuah property ICollecttion pada class CategoryDto seperti berikut.

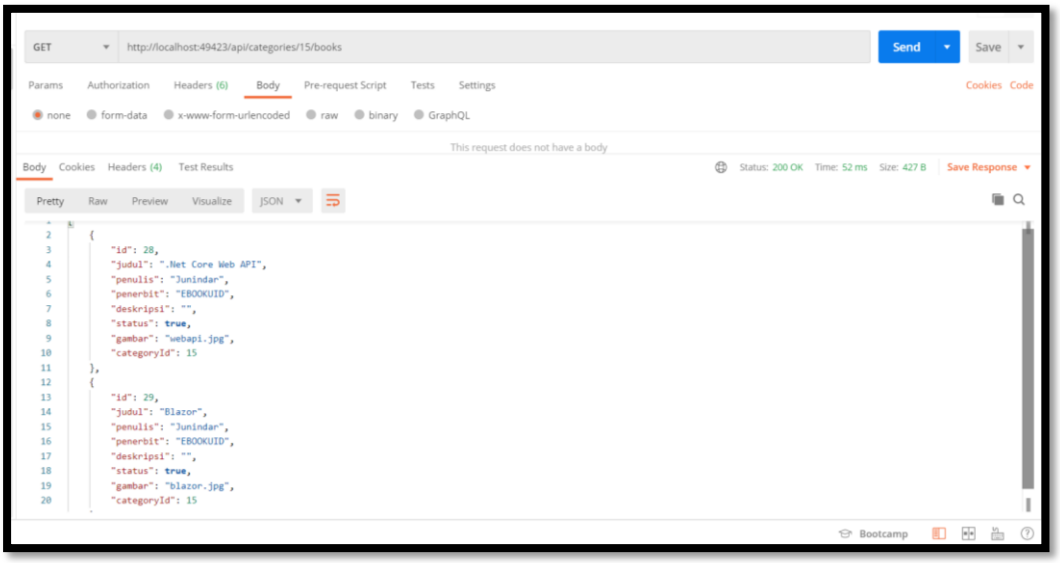
```
public ICollection<BookDto> Books { get; set; }= new List<BookDto>();
```



Untuk melakukan percobaan, ketikkan data json seperti berikut. Pada diatas kita akan menambahkan 1 buah data category dan 2 buah data buku. Lalu tekan button Send.



Gambar diatas merupakan hasil dari request yang telah kita lakukan. Pada Category dapat kita lihat ID nya adalah 15, sedangkan untuk buku adalah masing-masing 28 dan 29. Sedangkan untuk melihat apakah benar data-data tersebut ada pada database, kita dapat melakukan request seperti dibawah. /api/categories/15/books



Untuk latihan terakhir pada POST kita akan membuat sebuah proses pada web api untuk melakukan proses Insert untuk Collection. Yang artinya dalam sekali request kita dapat melakukan insert dengan lebih dari satu baris.

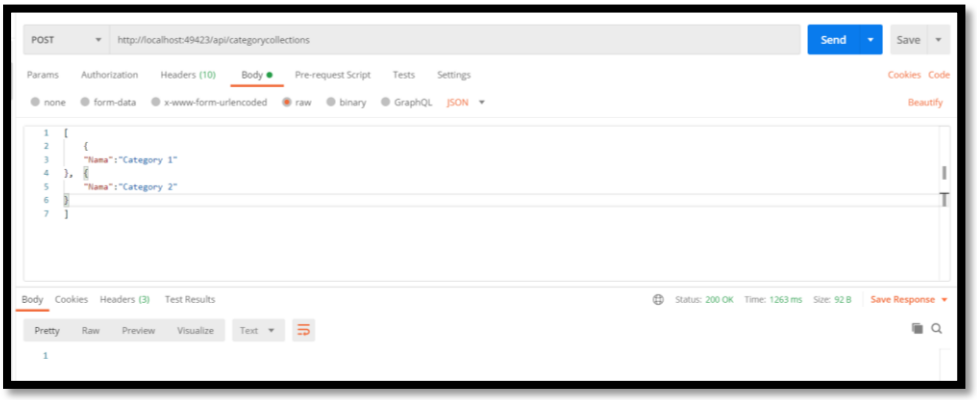
Untuk melakukan latihan ini, buat terlebih dahulu controller baru dengan nama "CategoryCollectionController". Dan untuk route pada controlle ini adalah seperti berikut [Route("api/categorycollections")]. Untuk sintaks detail nya dapat dilihat pada project lampiran. Disini akan dijelaskan hanya pada action nya saja, lalu ketikkan sintaks seperti dibawah ini.

```
[HttpPost]
public async Task<ActionResult<IEnumerable<CategoryDto>>>
       CreateMultiple(IEnumerable<CategoryDto> categoryColl)
{
       var categoryEntities = _mapper.Map<IEnumerable<Category>>(categoryColl);
       await _categoryRepository.InsertMultipleCategory(categoryEntities);

       return Ok();
}
```

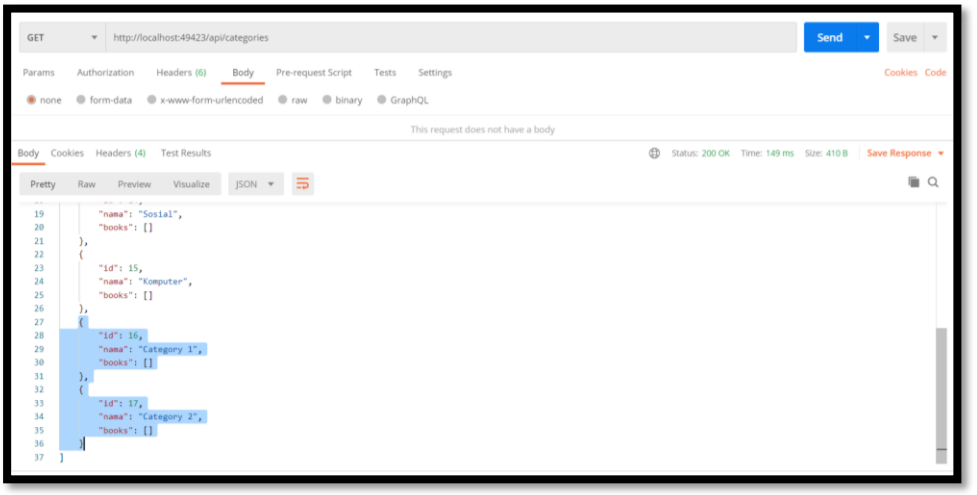
Pada method diatas, dapat dilihat parameternya adalah IEnumerable class CategoryDto. Yang pertama kali dilakukan oleh method ini adalah melakukan mapping menjadi IEnumerable class Category terlebih dahulu. Sebelumnya pastikan kita telah membuat sebuah method untuk melakukan proses multiple insert ini terlebih dahulu. Seperti pada sintaks dibawah ini.

```
public async Task InsertMultipleCategory(IEnumerable<Category> list)
{
        foreach (var itm in list)
        {
                _dbContext.Add(itm);
        }
        await _dbContext.SaveChangesAsync();
}
```

Selanjutnya dari action diatas, kita panggil method “InsertMultipleCategory“ ini.



Gambar diatas merupakan cara untuk melakukan request action multiple insert. Terdapat dua category yang akan ditambah kedalam database (Category 1 dan Category 2).

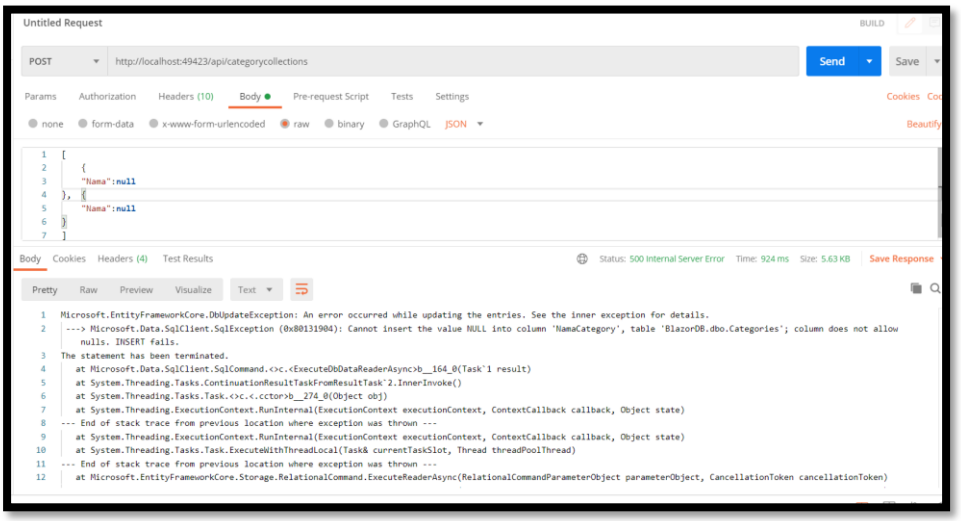


Untuk melihat hasilnya, apakah data-data yang telah kita tambah masuk kedalam table, dapat dilakukan seperti pada gambar diatas.

Disini kita telah membuat beberapa skenario untuk proses insert dari dengan menggunakan single hingga multiple data.

Selanjutnya kita akan membuat validasi dengan menggunakan atribut dari Data Annotations.

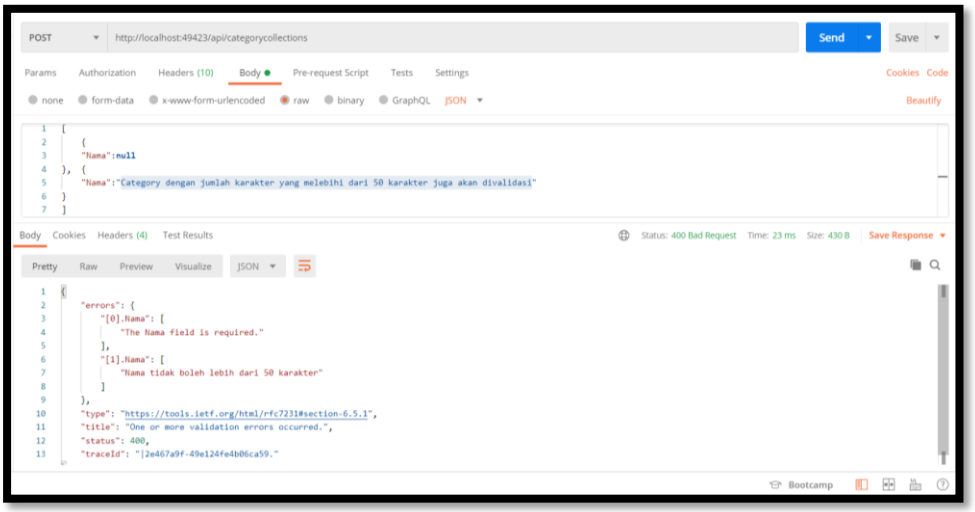
Sebelumnya coba lakukan request insert data category dengan data Nama=null, seperti pada gambar dibawah.



Dapat kita lihat pesan error yang didapat sangat tidak user friendly. Agar tidak terjadi hal seperti ini, kita dapat menggunakan Data Annotations pada property didalam class CategoryDto.

```
public class CategoryDto
{
      [Required]
      public int Id { get; set; }

      Required]
      [MaxLength(50, ErrorMessage = "Nama tidak boleh lebih dari 50 karakter")]
      public string Nama { get; set; }

      public ICollection<BookDto> Books { get; set; }= new List<BookDto>();
}
```

Dengan menggunakan Data Annotations, kita dapat menentukan property-property yang mana pada class yang tidak boleh sama dengan null. Lalu kita bisa menambahkan validasi untuk maksimal jumlah karakter yang bisa digunakan.

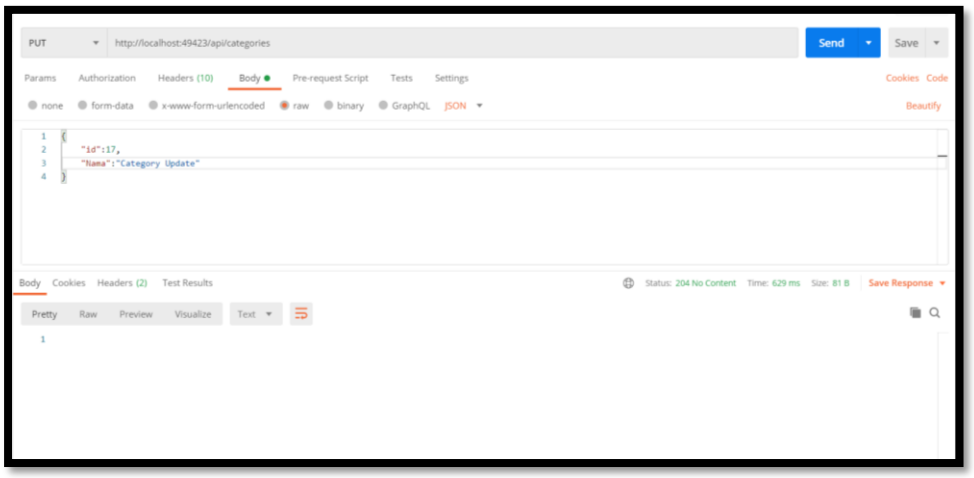


Pada gambar diatas, dapat dilihat dimana kita mencoba memasukkan dua buah category yang mana data pertama Nama sama dengan null, sedangkan yang kedua jumlah karakternya melebihi dari 50 karakter.

- PUT

HTTP PUT sering digunakan untuk melakukan proses update data. Cara penggunaan antara POST dan PUT pun tidak berbeda.

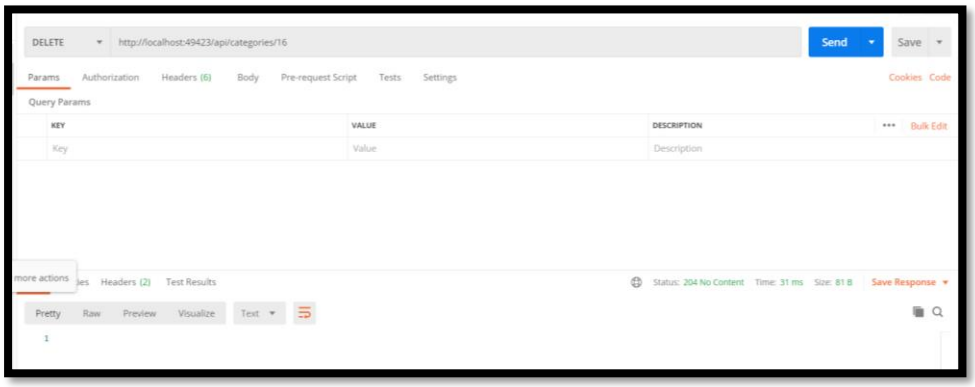
```
[HttpPut]
public async Task<ActionResult<CategoryDto>> UpdateCategory(CategoryDto category)
{
        var cat = await _categoryRepository.GetById(category.Id);

        if (cat == null)
        {
            return NotFound();
        }
        var categoryEntity = _mapper.Map<Category>(category);
        await _categoryRepository.UpdateCategory(categoryEntity);
        return NoContent();
}
```



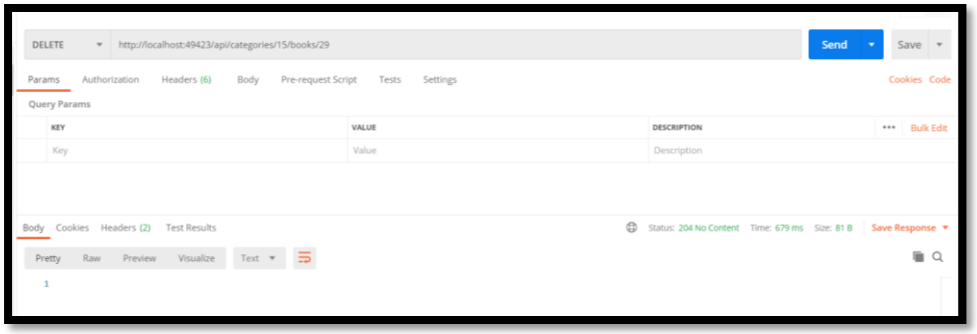
- DELETE

HTTP Delete dari namanya sudah menggambarkan fungsinya seperti untuk melakukan proses apa. Untuk menghapus data dari server dengan menggunakan webapi, makan kita akan gunakan Http Delete.

```
[HttpDelete("{categoryId}")]
public async Task<ActionResult<CategoryDto>> DeleteCategory(int categoryId)
{
      var cat = await _categoryRepository.GetById(categoryId);

       if (cat == null)
       {
           return NotFound();
       }
       await _categoryRepository.DeleteCategory(categoryId);


       return NoContent();
}
```



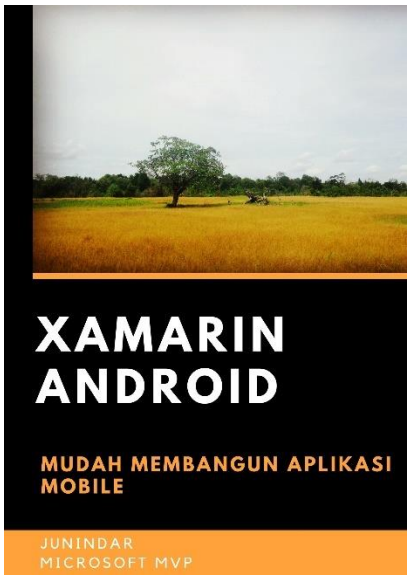

```
[HttpDelete]
[Route("api/categories/{categoryId}/books/{bookId}")]
public async Task<ActionResult<BookDto>> DeleteBookForCategory(int categoryId,
int bookId)
{
    var cat = await _categoryRepository.GetById(categoryId);

    if (cat == null)
    {
         return NotFound();
    }

    var book = await _bookRepository.GetBookById(bookId);

    if (book == null)
    {
        return NotFound();
    }

    await _bookRepository.Delete(bookId);
    return NoContent();
}
```


DELETE
http://localhost:49423/api/categories/15/books/29
Send
Save
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (2)
Test Results
Status: 204 No Content
Time: 679 ms
Size: 81 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1

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/2020/12/membuat-proses-crud-pada-web-api-aspnet.html

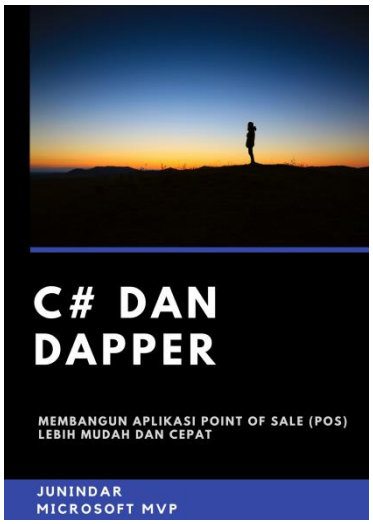
Referensi



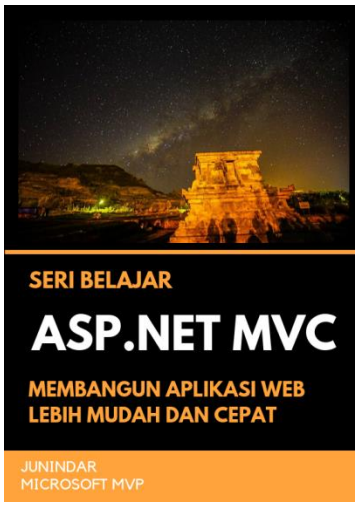
https://play.google.com/store/books/details?id=G4tFDgAAQBAJ

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

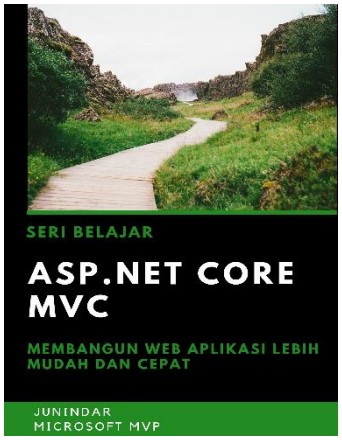
https://play.google.com/store/books/details/Junindar_Xamarin_Forms?id=6Wg-DwAAQBAJ



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".**