IlmuKomputer.Com
ikatlah ilmu dengan menuliskannya

Pengenalan WEB API (ASP.NET CORE)

Junindar, ST, MCPD, MOS, MCT, MVP

Lisensi Dokumen:
Copyright © 2003 IlmuKomputer.Com
*Seluruh dokumen di **IlmuKomputer.Com** dapat digunakan, dimodifikasi dan disebarkan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari **IlmuKomputer.Com**.*

junindar@gmail.com

http://junindar.blogspot.com

Abstrak

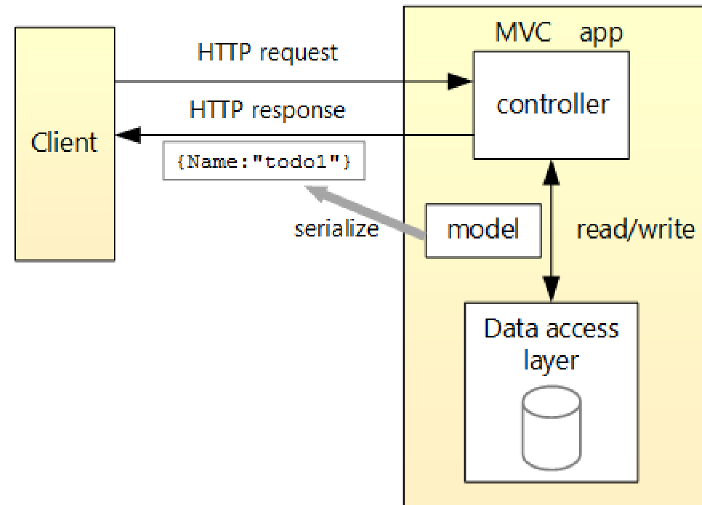
API adalah kepanjangan dari Application Programming Interface yang digunakan perangkat lunak untuk mengakses data, perangkat lunak server atau aplikasi lain dan telah ada selama beberapa waktu.

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.

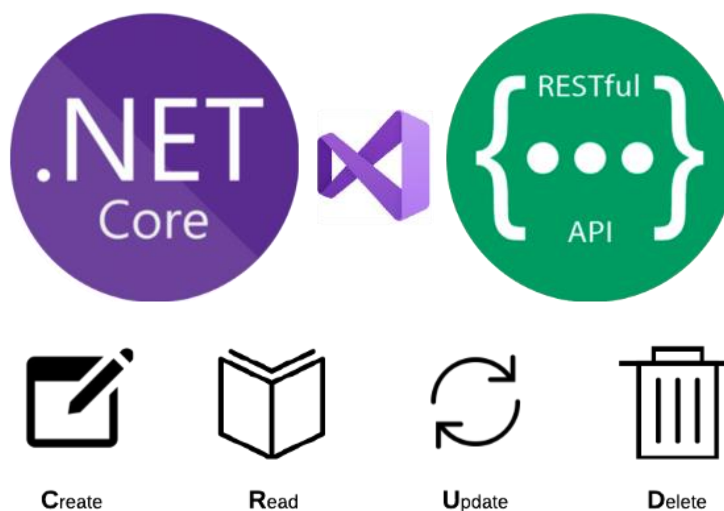
API dapat digunakan pada sistem berbasis web, sistem operasi, sistem basis data, dan perangkat keras komputer.

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.



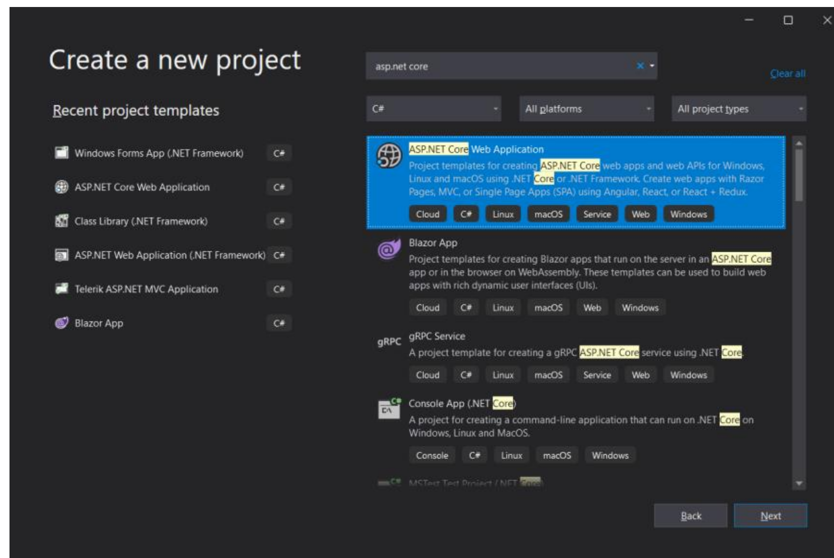
ASP.NET Core mendukung pembuatan layanan RESTful, juga dikenal sebagai Web API, dengan menggunakan C # sebagai bahasa pemogramannya. Untuk menangani request Web API menggunakan controller. Controller pada Web API adalah class yang berasal ControllerBase.



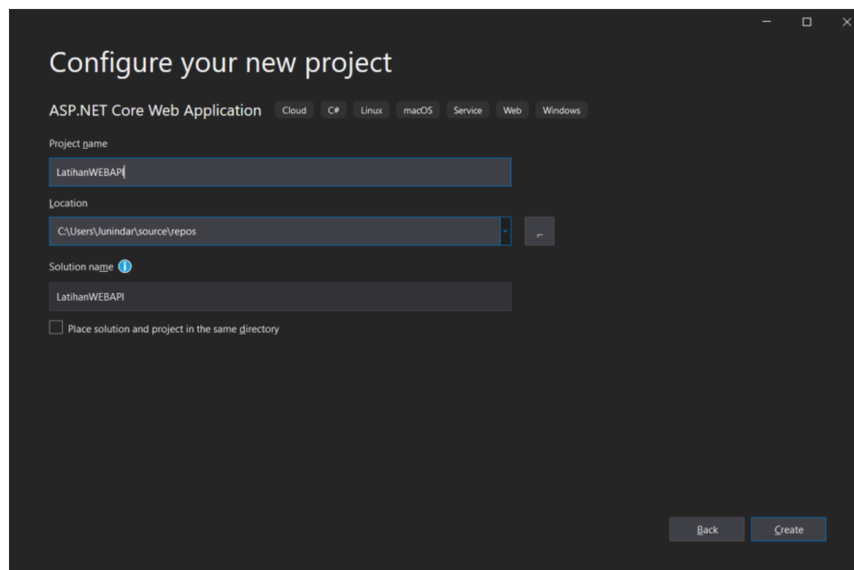
Pada artikel ini kita akan membuat project Web API dengan menggunakan ASP.NET Core.

Untuk memudahkan dalam memahami artikel ini, pertama kita buat terlebih dahulu sebuah project WEBAPI dari ASP.NET Core.

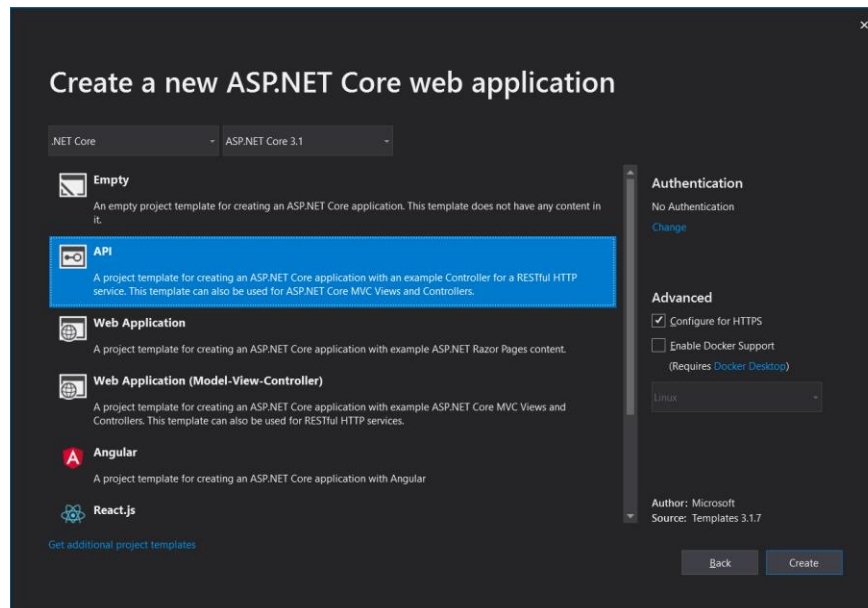
- Buka Visual Studio 2019 dan klik Create New Project
- Pada jendela Create New Project, pilih template ASP.NET Core Web Application dan klik Next button.



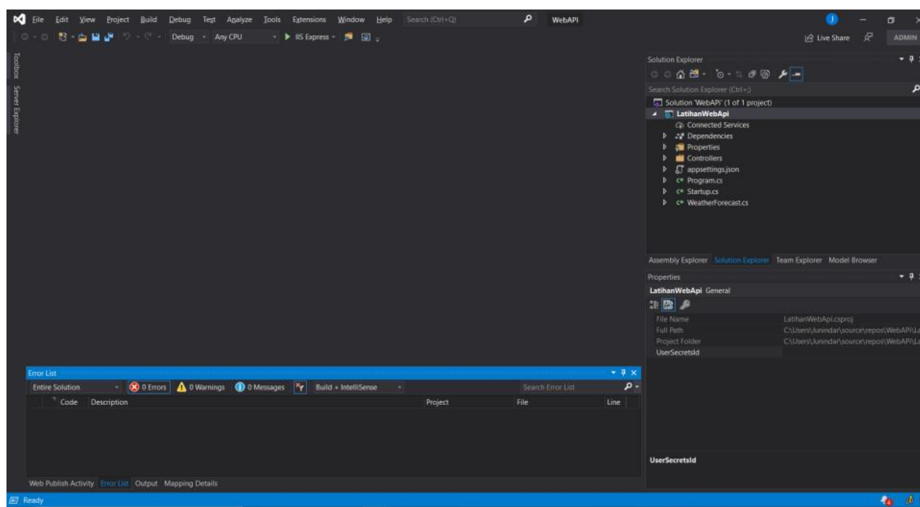
- Ketikkan nama project dan tentukan lokasi project ini akan disimpan. Klik Create button.



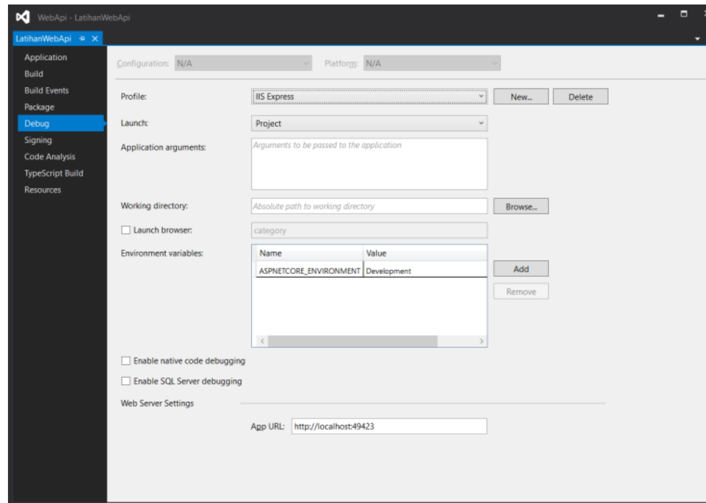
- Selanjutnya pilih template API, untuk latihan ini pastikan checkbox „"Configure for HTTPS" untuk tidak dipilih. Lalu klik Create button



- Setelah selesai dengan langkah-langkah diatas, Visual Studio akan membuat sebuah project dengan template yang telah kita pilih sebelumnya.



- Pada default project terdapat sebuah controller dan class "WeatherForecast", dua file tersebut tidak akan kita gunakan untuk latihan ini. Jadi sebaiknya kita hapus dua file ini dari project.
- Buka properties project, dengan cara klik kanan Project > Properties. Lalu pilih tab "Debug".

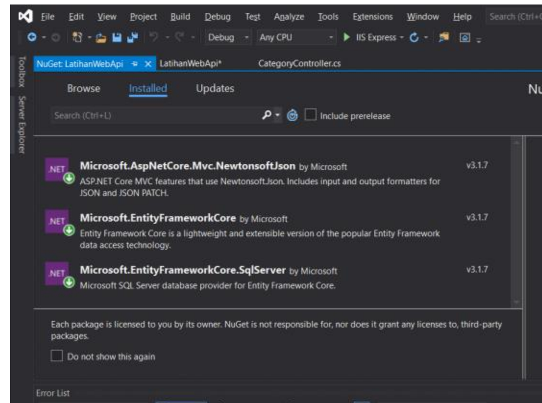


Pada Launch ganti value nya menjadi Project, lalu non aktifkan checkbox Launch browser. Dan terakhir ketikkan Api Url-nya. Dan coba jalankan project ini. Pastikan yang keluar adalah aplikasi console seperti dibawah ini.

```
E:\Program\Ilkom\Artikel Program\ilkom\WebApi\LatihanWebApi\bin\Debug\netcoreapp3.1\LatihanWebApi.exe
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:49423
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: E:\Program\Ilkom\Artikel Program\ilkom\WebApi\LatihanWebApi
```

Pada artikel ini tidak akan dijelaskan penggunaan EF Core dan proses migration database, sebaiknya pastikan telah membaca artikel-artikel sebelumnya seperti pada link berikut : http://junindar.blogspot.com/2019/10/aspnet-core-mvc-membangun-aplikasi-web.html

- Pastikan pada project ini telah di install beberapa NuGet package seperti dibawah ini.



- Untuk lebih memudahkan sebaiknya copy sintaks dari folder Data dan Entity. Dan pastikan sintaks pada Program.cs dan Startup.cs sama seperti pada project lampiran.

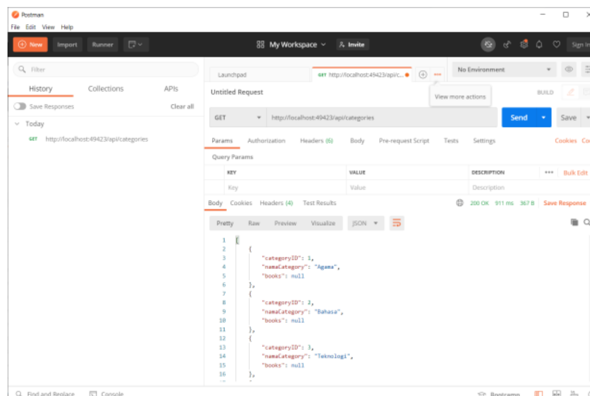
```
0 references | 0 changes | 0 authors, 0 changes
public class Program
{
    0 references | 0 changes | 0 authors, 0 changes
    public static void Main(string[] args)
    {
        var host = CreateHostBuilder(args).Build();

        using (var scope = host.Services.CreateScope())
        {
            var services = scope.ServiceProvider;

            var context = services.GetRequiredService<PustakaDbContext>();
            DbInitializer.Seed(context);


        }
        host.Run();
    }

    1 reference | 0 changes | 0 authors, 0 changes
    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

```
2 references | 0 changes | 0 authors, 0 changes
public class Startup
{
    0 references | 0 changes | 0 authors, 0 changes
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    2 references | 0 changes | 0 authors, 0 changes
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    0 references | 0 changes | 0 authors, 0 changes
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers().AddNewtonsoftJson(options =>
            options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore
        );
        services.AddDbContext<PustakaDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("pustakaConnection")));
        services.AddTransient<IBookRepository, BookRepository>();
        services.AddTransient<ICategoryRepository, CategoryRepository>();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    0 references | 0 changes | 0 authors, 0 changes
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}
```

- Tambahkan sebuah controller dengan nama CategoryController. Dan ganti Route pada contoller ini menjadi “`[Route("api/categories")]`“. Untuk lebih lengkapnya ketikkan sintaks seperti dibawah.

```
[ApiController]
[Route("api/categories")]
public class CategoryController : ControllerBase
{
     private readonly ICategoryRepository _categoryRepository;

     public CategoryController(ICategoryRepository categoryRepository)
     {
        _categoryRepository = categoryRepository;

     }

    [HttpGet()]
    public async Task<IActionResult> GetCategories()
    {
         var result = await _categoryRepository.GetAll();
         return new JsonResult(result);
    }
}
```

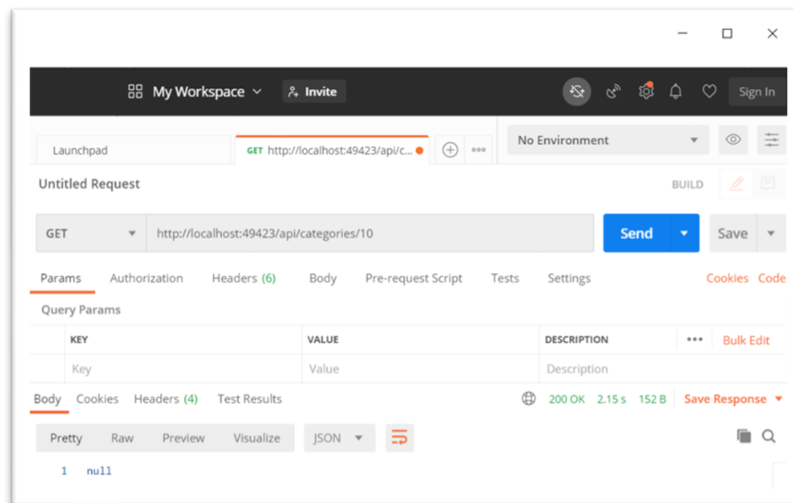
Controller diatas memiliki sebuah Action method dengan nama “GetCategories“. Dimana fungsinya untuk menampilkan semua data dari table “Categories“ dengan memanggil function “GetAll“ pada class CategoryRepository. Output dari action method ini adalah Json. Untuk melakukan pengetesan pastikan pada PC/laptop sudah di install Postman. Jalankan program dan buka aplikasi Postman. Pastikan http method nya adalah GET dan ketikkan url api berikut : http://localhost:49423/api/categories lalu klik button Send. Pastikan mendapatkan hasil seperti pada gambar dibawah ini.



- Setelah berhasil dengan proses diatas, akan kita lanjutkan dengan menampilkan untuk single data category. Tambakan sebuah action method seperti pada sintaks dibawah.

```
[HttpGet("{categoryId}")]
public async Task<IActionResult> GetCategory(int categoryId)
{
      var result = await _categoryRepository.GetById(categoryId);
      return new JsonResult(result);
}
```

Pastikan string template (categoryId) pada HttpGet saman dengan nama parameter pada Action method. Jalankan program dan lakukan pengetesan pada Postman. Sedangkan contoh url-nya seperti berikut: http://localhost:49423/api/categories/1 (angka 1 merupakan categoryId yang dicari). Lalu coba ganti nilai categoryId menjadi 10, dimana id ini tidak ada pada table, maka kita akan mendapatkan hasil seperti dibawah (null).



Status Code

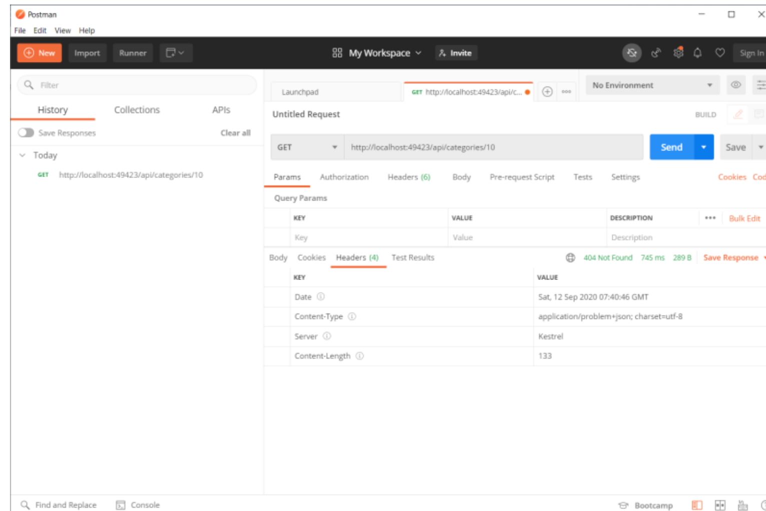
Status Code pada API sangat penting. Status yang dikirimkan oleh API untuk consumer akan memberikan informasi apakah API berjalan dengan baik atau tidak. Pada dua contoh action method diatas result yang dikirim oleh API adalah JSON result. Dimana Status Code yang dikirimkan selalu “200“. Berikut merupakan contoh beberapa Status Code yang sering digunakan.

Status Code	Description
Level 200 (Success)	
200	OK
201	Created
204	No Content
Level 400 (Client Mistake)	
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not found
405	Method not allowed
406	Not acceptable
409	Conflict
Level 500 (Server Mistakes)	
500	Internal Server Error

Sekarang kita akan mengganti action method diatas agar menggunakan status code yang benar. Ganti sintaks pada GetCategory menjadi seperti dibawah.

```
[HttpGet("{categoryId}")]
public async Task<IActionResult> GetCategory(int categoryId)
{
      var result = await _categoryRepository.GetById(categoryId);
      if (result == null)
      {
          return NotFound();
      }

      return Ok(result);
}
```

Pada sintaks diatas dapat dilihat, jika categoryId tidak ditemukan maka status code yang dikirimkan adalah 404. Seperti pada gambar dibawah ini.



Membuat Outer Facing Model

Pada latihan diatas, kita masih menggunakan model dari Entity Framework untuk return dari request. Entity model merupakan representasi dari table yang ada pada database. Sebaiknya kita pisahkan antara model Entity Framework dan Outer Facing dan pada latihan ini kita akan pisahkan model-model tersebut. Ikuti langkah-langkah dibawah ini.

- Tambahkan sebuah folder dengan nama “Models“.
- Lalu dalam folder tersebut, tambahkan sebuah class dengan nama „“CategoryDto“ dan ketikkan sintaks seperti dibawah ini.

```
namespace LatihanWebApi.Models
{
    public class CategoryDto
    {
        public int Id { get; set; }
        public string Nama { get; set; }
    }
}
```

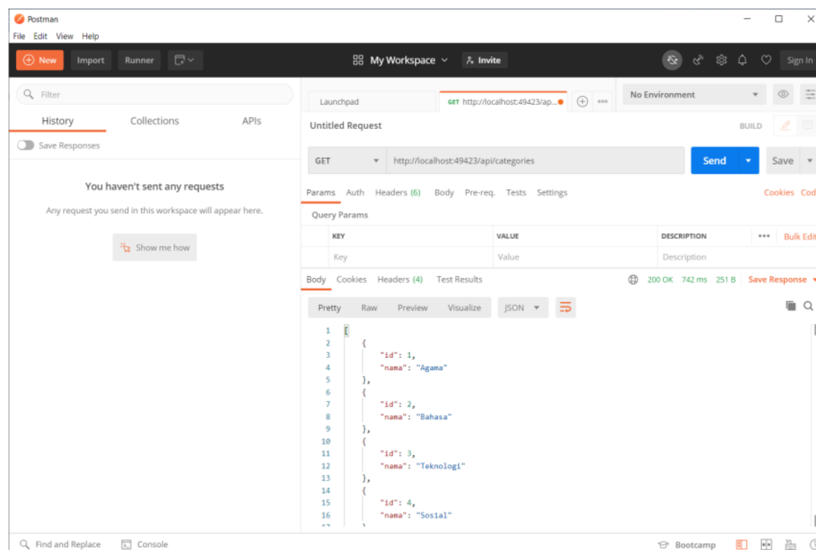
- Lalu untuk CategoryController, ganti sintaks pada GetCategories dan GetCategory, seperti dibawah ini.

```
[HttpGet()]
public async Task<IActionResult> GetCategories()
{
       var resultRepo = await _categoryRepository.GetAll();
       var result = resultRepo.Select(itm => new CategoryDto()
                {
                    Id = itm.CategoryID,
                    Nama = itm.NamaCategory}).ToList();

       return new JsonResult(result);
}
```

```
[HttpGet("{categoryId}")]
public async Task<IActionResult> GetCategory(int categoryId)
{
     var resultRepo = await _categoryRepository.GetById(categoryId);

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

     var result = new CategoryDto {Id = resultRepo.CategoryID,
      Nama = resultRepo.NamaCategory};

     return Ok(result);
}
```

Jika dilihat pada sintaks-sintaks diatas, tidak banyak perubahan yang dilakukan. Hasil dari pencarian pada service akan dimasukkan kedalam object baru sebagai hasil dari request.

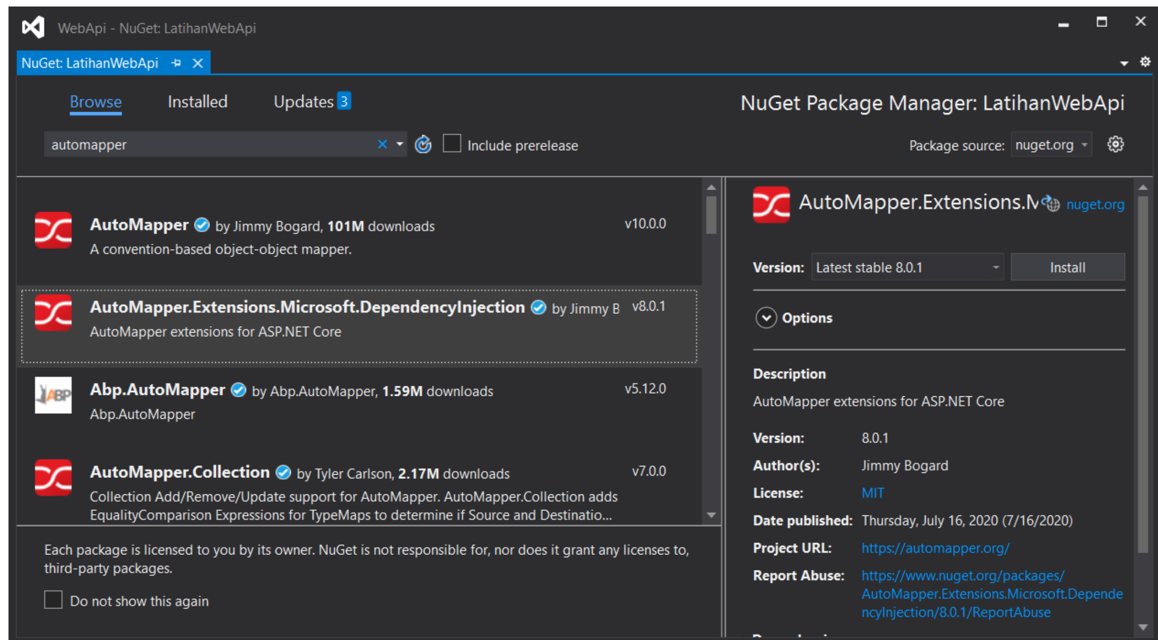
Jalankan program dan pastikan mendapatkan hasil seperti dibawah.



AutoMapper

Pada latihan sebelumnya, untuk memasukkan data pada CategoryDto kita masih menggunakan manual mapping. AutoMapper memudahan kita dalam melakukan mapping antara object. Untuk menggunakan AutoMapper ikuti langkah-langkah dibawah ini.

- Buka NuGet Package Manager, cari dan install “AutoMapper.Extensions.Microsoft.DependencyInjection“ .



- Setelah selesai, buka Startup.cs dan tambahkan sintaks berikut pada ConfigureService. Hal ini dilakukan untuk register service AutoMapper pada container.

```
services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());
```

- Tambahkan sebuah folder dengan nama Profiles, pada folder tersebut juga tambahkan sebuah class “CategoryProfile“, dan untuk melakukan konfigurasi AutoMapper ketikkan sintaksnya seperti berikut.

```
public class CategoryProfile : Profile
{
        public CategoryProfile()
        {
            CreateMap<Category,
            CategoryDto>().ForMember(dest=>dest.Id,opt=>opt.MapFrom(src=>src.CategoryID))
                .ForMember(dest => dest.Nama, opt => opt.MapFrom(src => src.NamaCategory)
            );
        }
}
```

Dikarenakan nama property antara Entity Model dan Outer Facing Model. tidak sama, maka kita perlu mengindentifikasi untuk setiap property dari masing-masing Model. Seperti property Id pada Outer Facing akan di-mapping dengan property CategoryID pada Entity Model.

- Selanjutnya Buka CategoryController dan tambahkan sebuah private readonly, dan ganti sintak constructor seperti dibawah.

```
private readonly IMapper _mapper;
public CategoryController(ICategoryRepository categoryRepository,IMapper mapper)
{
      _categoryRepository = categoryRepository;
      _mapper = mapper;
}
```

- Dan terakhir kita akan melakukan mapping object dari Entity (Category) ke CategoryDto, seperti dibawah.

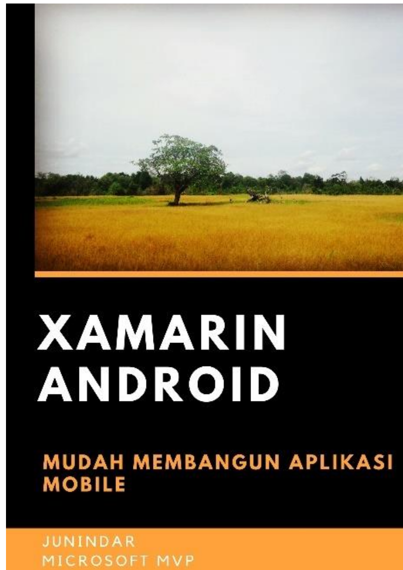
```
[HttpGet()]
public async Task<IActionResult> GetCategories()
{

    var resultRepo = await _categoryRepository.GetAll();
    return Ok(_mapper.Map<IEnumerable<CategoryDto>>(resultRepo));

}


[HttpGet("{categoryId}")]
public async Task<IActionResult> GetCategory(int categoryId)
{
     var resultRepo = await _categoryRepository.GetById(categoryId);

     if (resultRepo == null)
     {
                return NotFound();
     }
     return Ok(_mapper.Map<CategoryDto>(resultRepo));
}
```

Dengan menggunakan AutoMapper kita tidak perlu lagi melakukan mapping secara manual pada setiap hasil pencarian dari service.

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/2020/09/pengenalan-web-api-aspnet-core.html

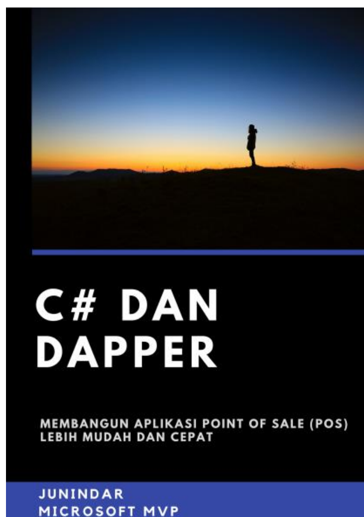
Referensi



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

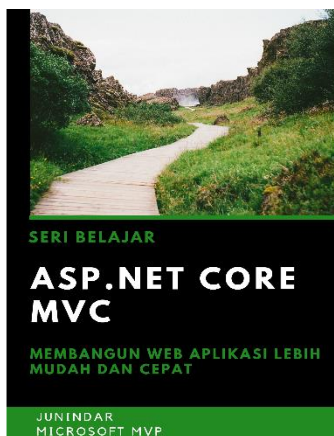
https://play.google.com/store/books/details?id=VSLiDQAAQBAJ

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".**