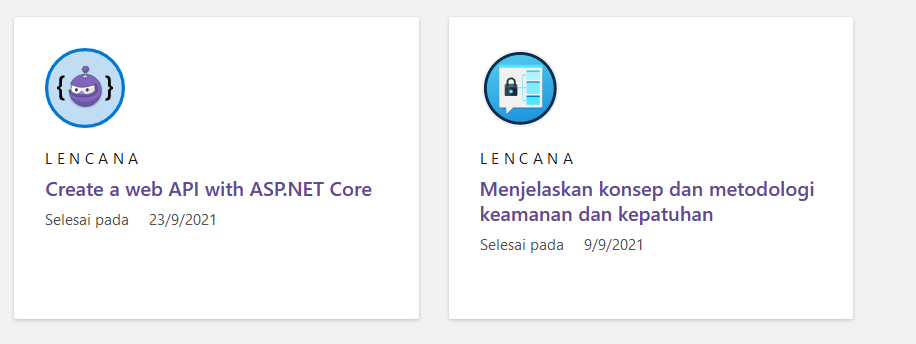
Renaldi Soeryadi, 72190301

Screenshoot Lencana (badge)



Ringkasan

Dalam metode tradisional, sebuah web server berkomunikasi dengan browser yang kita gunakan menggunakan sintaks HTML, CSS, dan Javascript yang diketik dalam program yang kemudian dieksekusi. Akan tetapi, dalam cara yang sama, sebuah web server berkomunikasi dengan klien-klien yang berbeda, seperti perangkat mobile, web server yang lain, dan lain-lain menggunakan layanan web yang dihubungkan oleh API dengan menggunakan format data JSON atau XML, yang berisikan data-data yang dikirimkan melalui HTTP menuju server penerima. API biasanya digunakan dalam Single-Page Applications yang melakukan hampir semua logika UI dalam web browser dan berkomunikasi dengan web server dengan API.

Salah satu alur pembangunan API sendiri adalah REST (Representational State Transfer), yang merupakan sebuah arsitektur pembangunan layanan web, dimana metode REST menggunakan HTTP yang sama dengan HTTP yang digunakan dalam web browser untuk menerima data atau mengirimkan data ke web server untuk diproses. Bahasa yang dipakai yang berkaitan dengan data, yang sering disebut dengan metode HTTP, ada 5, yaitu :

1. GET = Bahasa yang digunakan untuk menerima data dari layanan web.
2. POST = Bahasa yang digunakan untuk mengirimkan data ke layanan web.
3. PUT = Bahasa yang digunakan untuk mengubah data-data yang ada dalam layanan web.
4. PATCH = Bahasa yang digunakan untuk mengubah data dalam layanan web yang menggunakan beberapa instruksi tentang apa yang seharusnya diubah.
5. DELETE = Bahasa yang digunakan untuk menghapus data dalam web service.

Sedangkan ada satu lagi turunan dalam API, yaitu RESTful API, dimana Teknik RESTful API ini menggunakan Teknik REST tadi, dengan menggunakan URI standar, metode HTTP, dan format data seperti JSON atau XML.

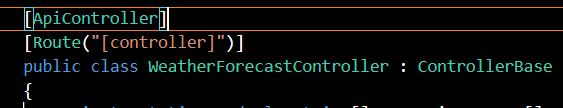
Ada beberapa manfaat dari penggunaan API dalam ASP.NET Core, diantaranya :

1. Serialisasi yang mudah = Dalam pembentukan API menggunakan ASP.NET, semua titik temu (*endpoints*) diserialisasikan secdara otomatis menggunakan format JSON, sehingga tidak perlu menginfigurasikan lagi, kecuali untuk kustomisasi yang memerlukan persyaratan unik.
2. Autentikasi dan Otorisasi = Semua API yang digunakan di-*support* menggunakan JSON Web Token, yang memungkinkan fleksibilitas untuk menentukan peraturan control akses data.
3. Routing sepanjang kode anda = ASP.NET memungkinkan anda untuk menentukan Bahasa-bahasa apa yang digunakan dalam kode anda, dengan menggunakan atribut yang dieksekusi yang langsung diikat pada parameter metode yang digunakan.
4. HTTPS sejak awal = HTTPS sendiri menggunakan *end-to-end encryption*, untuk menyediakan privasi dan memungkinkan pemanggilan API yang dapat dipercaya, karena tidak dapat diubah dan tidak dapat diganggu. ASP.NET secara otomatis menyediakan sebuah sertifikat tes dan dapat diimpor dengan mudah untuk menyambungkan HTTPS lokal.
5. Membagikan kode dan pengetahuan dengan aplikasi .NET yang tersedia dalam beberapa *platform*.

Setelah API selesai dibuat dan dikembangkan, tentu harus ada pengetesan untuk memasatikan API yang digunakan itu dapat digunakan. Aplikasi .NET sudah menyediakan layanan .NET HTTP REPL (HTTP Read-Eval-Print Loop), yang memungkinkan pengguna untuk membangun lingkungan CLI yang interaktif.

Dalam Web API menggunakan ASP.NET, kita perlu mengetahui terlebih dahulu dua jenis kelas dalam Pengontrol (*Controller*) API, yaitu BaseClass yang merupakan ControllerBase dan API Controller.

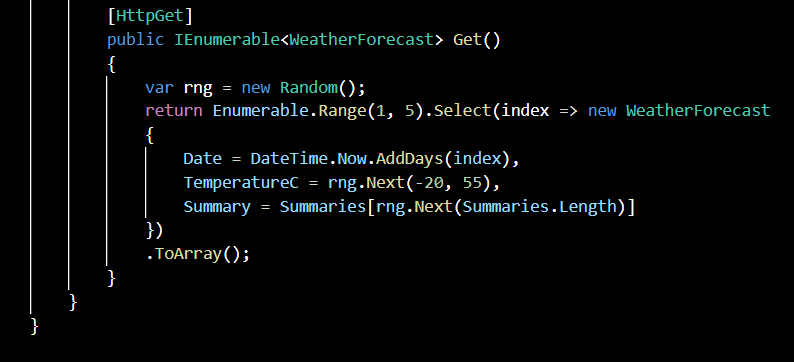
ApiController



ControllerBase

ControllerBase merupakan BaseClass, dimana nantinya ControllerBase ini akan digunakan untuk mewariskan (*inherits*) banyak fungsi standar untuk mengendalikan HTTP request yang berkaitan dengan API dalam *controller* API kita, seperti GET, POST, PUT, PATCH, dan DELETE. Sedangkan ApiController menyediakan perilaku yang memudahkan kita untuk membangun web API, sedangkan Route yang terletak dibawah ApiController bertujuan untuk membuat rute pemanggilan ketika halaman tersebut dipanggil.

Ketika halaman API dipanggil, maka system akan menjalankan atribut HttpGet, dimana semua data nantinya diambil dan dikembalikan dalam bentuk keluaran kepada pembaca.



Nah, dalam logika penangkapan data tersebut, kita menggunakan IEnumerable, dengan maksud agar system dapat menunda evaluasi yang ada dalam logika didalamnya sehingga program dapat di-*compile* dan dapat diekeskusi dengan optimal. Nantinya, IEnumerable menggunakan kelas WeatherForecast untuk dapat menangkap data yang sudah ada. Sedangkan Kelas Random pada var rng menandakan bahwa nantinya system akan membuat serangkaian angka acak, dimana nantinya akan memproduksi sequence yang memenuhi persyaratan statistic untuk logika yang ada dibawahnya, dimana nanti kode yang dibawahnya menandakan berapa banyak jumlah data yang nantinya ditampilkan pada kode-kode program dibawahnya, dimana nantinya semua hasil yang dikeluarkan dirangkum menjadi satu dalam bentuk Array menggunakan tag .toArray().

Nanti, hasilnya akan tampak sebagai berikut.

**[**

**{**

**"date": "2021-09-24T10:46:04.072647+07:00",**

**"temperatureC": -5,**

**"temperatureF": 24,**

**"summary": "Bracing"**

**},**

**{**

**"date": "2021-09-25T10:46:04.0726527+07:00",**

**"temperatureC": 43,**

**"temperatureF": 109,**

**"summary": "Sweltering"**

**},**

**{**

**"date": "2021-09-26T10:46:04.0726532+07:00",**

**"temperatureC": 45,**

**"temperatureF": 112,**

**"summary": "Warm"**

**},**

**{**

**"date": "2021-09-27T10:46:04.0726534+07:00",**

**"temperatureC": -7,**

**"temperatureF": 20,**

**"summary": "Mild"**

**},**

**{**

**"date": "2021-09-28T10:46:04.0726536+07:00",**

**"temperatureC": 45,**

**"temperatureF": 112,**

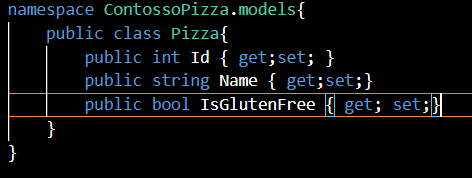
**"summary": "Balmy"**

**}**

**]**

Nah, untuk menambahkan data, kita dapat menggunakan skema MVC (Models, View, Controller), dimana Models menandakan relasi antara system dengan database nya. Tetapi, karena kita tidak menggunakan Database, kita dapat mendefinisikan atribut-atribut yang nanti dipasangkan pada Kelas Pizza berikutnya, berikut dengan logika pemrogramannya.

Pertama-tama, kita buat dahulu folder Models, dimana nanti folder tersebut akan terdapat folder pizza.cs yang berisikan deklarasi atribut-atribut seperti berikut.



Nanti, setelah deklarasi atribut selesai dibuat, maka kita perlu membuat data, membuat logika penangkapan data, serta membuat logika pemrograman lain yang berhubungan dengan CRUD data, seperti menambahkan data (Add), mengganti data (Update), dan menghapus data(Delete).

Untuk itu, kita dapat membuat suatu folder baru yang Bernama Services, dimana nantinya folder ini akan menampung logika modifikasi data yang nantinya dilakukan. Untuk itu, kita dapat mendeklarasikan proses penangkapan datanya dulu seperti ini.

namespace ContossoPizza.Services {

    public static class PizzaService{

        static List<Pizza> Pizzas { get;}

        static int nextId = 3;

        static PizzaService(){

            Pizzas = new List<Pizza>{

                new Pizza { Id = 1, Name = "Classic Italian", IsGlutenFree = false},

                new Pizza { Id = 2, Name = "Veggie", IsGlutenFree = true}

                };

            }

        public static List<Pizza> GetAll() => Pizzas;

        public static Pizza Get(int id) => Pizzas.FirstOrDefault(p => p.Id == id);

Lalu, kita dapat menambahkan pula logika CUD (Create, Update, Delete) data, seperti berikut.

public static void Add(Pizza pizza){

            pizza.Id = nextId++;

            Pizzas.Add(pizza);

        }

        public static void Delete(int id)

        {

            var pizza = Get(id);

            if(pizza is null)

                return;

            Pizzas.Remove(pizza);

        }

        public static void Update (Pizza pizza){

            var index = Pizzas.FindIndex(p => p.Id == pizza.Id);

            if(index == -1){

                return;

            }

            Pizzas[index] = pizza;

        }

    }

}

Kalau Sudah, kita dapat membuat Controllernya.

*Controller* ini nantinya akan menghubungkan bagian *Models* dengan bagian *View,* dimana nantinya pada bagian Controller ini berisikan logika-logika pemrosesan data, sebelum dikirim ke Database ataupun ditampilkan ke layar.

Nah, setelah kelas PizzaController dibuat, mari kita buat logika CRUD datanya.

1. Get Data

[HttpGet]

        public ActionResult<List<Pizza>> GetAll() => PizzaService.GetAll();

        //GET A PIZZA

        [HttpGet("{id}")]

        public ActionResult<Pizza> Get(int id){

            var pizza = PizzaService.Get(id);

            if(pizza == null){

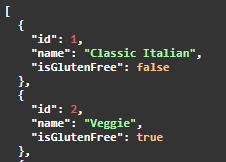
                return NotFound();

            }

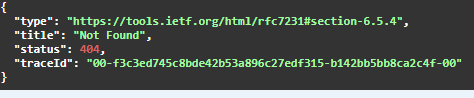
            return pizza;

        }

Pada contoh diatas, saya memiliki dua buah metode, yaitu metode GetAll yang mendapatkan semua data yang tersedia dengan menggunakan kelas ActionResult<>, dimana nantinya metode tersebut akan mengembalikan tipe dalam semua *controller* dalam MVC yang sudah dibuat. Nanti, untuk metode GetAll, hasilnya akan tampak sebagai berikut.



Sedangkan metode yang dibawahnya, metode Get, adalah metode untuk menangkap data dengan menggunakan ID yang sudah kita set sebelumnya. Nanti, apabila id yang diminta tersedia, maka system akan menampilkan datanya. Sedangkan kalua tidak tersedia, maka nanti system akan mengembalikan metode NotFound(), dimana hasilnya nanti akan tampak sebagai berikut.



Dari gambar diatas, kita dapat menemukan kode 404 dalam Statusnya, yang merupakan kode untuk menampilkan parameter yang tidak ada.

1. Add Data

[HttpPost]

        public IActionResult Create(Pizza pizza) {

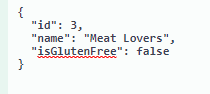
            PizzaService.Add(pizza);

            return CreatedAtAction(nameof(Create), new { id = pizza.Id }, pizza);

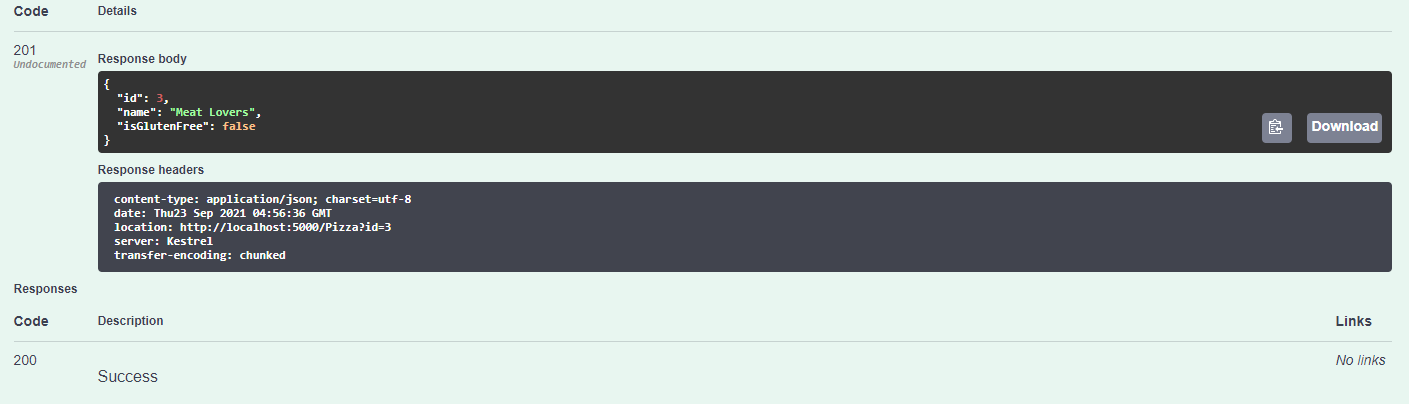
        }

Metode diatas adalah metode yang digunakan untuk membuat daftar Pizza Baru, dimana nantinya metode tersebut akan menjalankan metode Add pada PizzaService yang sudah kita buat sebelumnya. Lalu, nantinya, metode tersebut akan mengembalikan metode CreatedAtAction, dimana nantinya metode tersebut akan menghasilkan Status 201Created, dimana nantinya akan meghasilkan tanda bahwa data berhasil dimasukkan.

Contohnya adalah, saya memiliki sebuah data yang ingin saya tambahkan dibawah ini.



Nantinya, system akan menjalankan metode tersebut sehingga nanti ada kode 201 *undocumented* pada samping data yang sudah ditambahkan tersebut.



Nah, kode 201 *undocumented* ini merupakan hasil dari metode CreatedAtAction tadi. Tetapi, sebenarnya, data sudah ditambahkan kedalam system., ditandai dengan adanya kode 200 yang menunjukkan bahwa proses tersebut sukses dilakukan.

1. Update Data

[HttpPut("{id}")]

        public IActionResult Update(int id, Pizza pizza) {

            if (id != pizza.Id)

                return BadRequest();

            var existingPizza = PizzaService.Get(id);

            if(existingPizza is null)

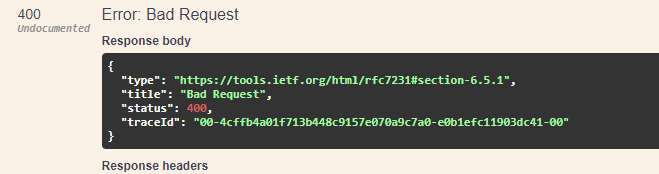
                return NotFound();

            PizzaService.Update(pizza);

            return NoContent();

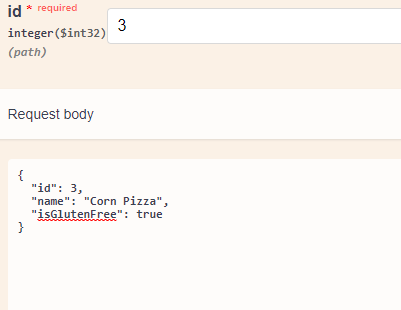
    }

Metode diatas nantinya akan menjalankan proses pengubahan data, dimana nantinya akan ada dua percanabgan. Nanti, apabila nomor Id pada Pizza nya tidak cocok, maka nanti akan menjalankan proses BadRequest(), dimana nanti akan menampilkan status 400.



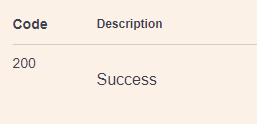
Lalu, pada percabangan berikutnya, apabila data pada Pizza yang ada tidak tersedia, maka nanti akan menjalankan metode NotFound(), dimana nantinya system akan membuat kode 404 yang menandakan bahwa data tidak tersedia.

Sisanya, kalau data ada, maka akan diubah sesuai dengan deskripsi yang kita ubah dalam kelas PizzaService bagian Update, seperti pada contoh dibawah ini.



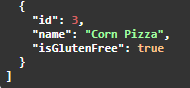
|

V



|

V



(Sebelumnya merupakan Meat Lovers).

1. Delete Data

[HttpDelete("{id}")]

        public IActionResult Delete(int id) {

            var pizza = PizzaService.Get(id);

            if (pizza is null)

                return NotFound();

            PizzaService.Delete(id);

            return NoContent();

        }

Metode diatas nantinya menjalankan proses penghapusan data, dimana system akan mengecek ID Pizza yang ada. Apabila datanya tidak tersedia, maka nanti akan menjalankan metode NotFound(), dimana nantinya system akan membuat kode 404 yang menandakan bahwa data tidak tersedia.

Sisanya, nanti system akan menjalankan proses Delete pada PizzaService, sebelum mengembalikan NoContent, yang merupakan proses penghapusan data yang diubah dan diubah menjadi data final.