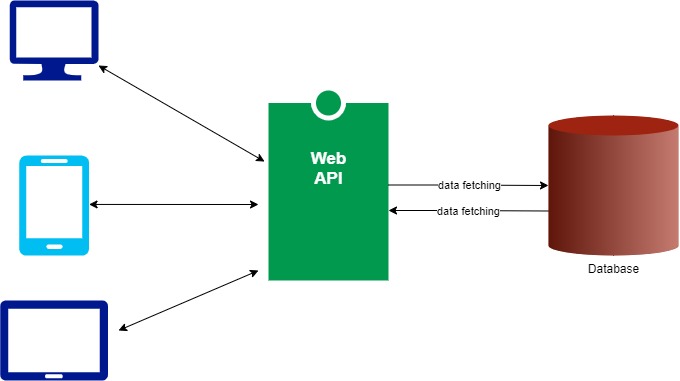
**What Is Web API:**

[](https://blogger.googleusercontent.com/img/a/AVvXsEiUi3Atnp9ap6QpKMI7SJHYvQbsvi2UjU0gEvNhDn6XJ8seVTE4n_JdVggTD1K_neZ6dnYq6baFz-4EWyQbCp2wR8xPt3zgLy_w32komPm_mGsx95jvVjHexRb-gaF9xqxB2p-sxq3bG63ZFRVZHZZO_fmZnpmcgrww7uBrwdpcL3sFlnXShHvLDZZG=s680)

Web API is a framework for building HTTP services that can be accessed from any client like browser, mobile devices, desktop apps.

In simple terminology API(Application Programming Interface) means an interface module that contains a programming function that can be requested via HTTP calls to save or fetch the data for their respective clients.

Some of the key characteristics of API:

* Supports HTTP verbs like 'GET', 'POST', 'PUT', 'DELETE', etc.
* Supports default responses like 'XML' and 'JSON'. Also can define custom responses.
* Supports self-hosting or individual hosting, so that all different kinds of apps can consume it.
* Authentication and Authorization are easy to implement.
* The ideal platform to build REST full services.

**Create A .NET6 Web API Application:**

Let's create a .Net6 Web API sample application to accomplish our demo. We can use either Visual Studio 2022 or Visual Studio Code(using .NET CLI commands) to create any.Net6 application. For this demo, I'm using the 'Visual Studio Code'(using the .NET CLI command) editor.

CLI command  
dotnet new webapi -o Your\_Project\_Name

Now let's explore default services or middleware that are in 'Program.cs':

***Services in Program.cs:***

1. builder.Services.AddControllers();
2. builder.Services.AddEndpointsApiExplorer();
3. builder.Services.AddSwaggerGen();

* (Line:1) The 'AddControllers' service for API Controllers. This method will not work for the Views or Pages.
* (Line: 2) The 'AddEndpointsApiExplorer' service for endpoint metadata.
* (Line: 3) The 'AddSwagerGen' service for the Swagger.

***Middlewares in Program.cs:***

1. **if** (app.Environment.IsDevelopment())
2. {
3. app.UseSwagger();
4. apps . UseSwaggerUI ();
5. }
6. app.UseHttpsRedirection();
7. app.UseAuthorization();
8. app.MapControllers();

* (Line: 3-4) The swagger middleware to load the swagger page.
* (Line: 6) The 'UseHttpsRedirection' middleware to redirect the 'HTTP' request to 'HTTPS'.
* (Line: 7) The user authorization middleware.
* (Line: 8) The 'MapControllers' is an endpoint middleware that can direct the request to API's controllers.

**SQL Query To Create A Sample Table:**

Run the below command to create a sample table.

CREATE TABLE [dbo].[Cake] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[Name] VARCHAR (MAX) NULL,

[Price] DECIMAL (18, 2) NULL,

[Description] VARCHAR (MAX) NULL

);

**Entity Framework Core:**

Entity Framework Core is an Object/Relational Mapping(ORM) framework. EF Core makes database communication more fluent and easy. The 'DatabaseContext' class acts as a database from our c# code, it will contain all registered tables as 'DbSet<TEntity>'(TEntity is any POCO class of a table).

[](https://blogger.googleusercontent.com/img/a/AVvXsEgUhij8VJELN6Yxm0H1dXb88kfbw1-Jm6DUF7btxdDbvgRzl6UfNkSVD5FOUILycl758gDiQOQY3EZ67d5Iaf8Ja7kdBU_TccKfHjYCVBFCFRhJ8208T-nWxmh2sfdIV3-fFlqUDju5JLyH2ZwggRwUTK64yR8KKwHxBPSyT99XRIxGkE2Q3g8IpMgQ=s621)

In this demo, we are going to implement the 'Code First With Existing Database' technique. Creating the classes for an existing database is known as 'Code First With Existing Database'.

**Install Entity Framework Core NuGet:**

Now install the Entity Framework Core NuGet.

Package Manager Command  
Install-Package Microsoft.EntityFrameworkCore -Version 6.0.1

.Net CLI Command  
dotnet add package Microsoft.EntityFrameworkCore --version 6.0.1

Now install SQLServer which is Entity Framework Core dependent library.

Package Manager Command  
Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 6.0.1

Package Manager Command  
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 6.0.1

**Setup Entity Framework Core**

**DatabaseContext:**

Let's create a class that represents our table. So let's create folders like 'Data' and subfolder like 'Entities' and then add a class like 'Cake.cs'.

***Data/Entities/Cake.cs:***

1. namespace Dot6.API.Crud.Data.Entities;
3. **public** **class** Cake
4. {
5. **public** int Id { **get**; set; }
6. **public** **string** Name { **get**; set; }
7. **public** **decimal** Price { **get**; set; }
8. **public** **string** Description { **get**; set; }
9. }

To manage or control all the table classes we have to create DatabaseContext class. So let's create our context class like 'MyWorldDbContext.cs' in the 'Data' folder.

**Data/MyWorldDbContext.cs:**

1. using Dot6.API.Crud.Data.Entities;
2. using Microsoft.EntityFrameworkCore;
4. namespace Dot6.API.Crud.Data;
6. **public** **class** MyWorldDbContext : DbContext
7. {
8. **public** MyWorldDbContext(DbContextOptions<MyWorldDbContext> options) : **base**(options)
9. {
11. }
12. **public** DbSet<Cake> Cake { **get**; set; }
13. }

* (Line: 6) The 'Microsoft.EntityFrameworkCore.DbContext' needs to be inherited by our 'MyWorldDbcontext' to act as a Database context class.
* (Line: 8) The 'Microsoft.EntityFrameworkCore.DbContextOptions' is instance of options that we are going to register in 'Program.cs' like 'ConnectString', 'DatabaseProvider', etc.
* (Line: 12) All our table classes must be registered inside of our Database context class with 'DbSet<T>' so that the entity framework core can communicate with the table of the database.

In the 'appsettings.Development.json' file add a connection string.

***appsettings.Development.json:***

"ConnectionStrings":{

"MyWorldDbConnection":"your\_connection"

}

Register the Database Context service in the 'Program.cs' file for dependency injection.

Program*.cs:*

builder.Services.AddDbContext**<MyWorldDbContext>**(options =>

{

options.UseSqlServer(builder.Configuration.GetConnectionString("MyWorldDbConnection"));

});

**Create A Sample API Controller:**

In Web APIs controllers are the entry point for the HTTP requests. The 'Action' methods are logical units in the controller which get executed and return the response per request.

Now create a sample API controller like 'CakeController.cs' inside of the 'Controllers' folder.

**Controllers/CakeController.cs:**

1. using Dot6.API.Crud.Data;
2. using Microsoft.AspNetCore.Mvc;
4. namespace Dot6.API.Crud.Controllers;
6. [ApiController]
7. [Route("[controller]")]
8. **public** **class** CakeController: ControllerBase
9. {
10. **private** **readonly** MyWorldDbContext \_myWorldDbContext;
11. **public** CakeController(MyWorldDbContext myWorldDbContext)
12. {
13. \_myWorldDbContext = myWorldDbContext;
14. }
15. }

* (Line: 6) Decorated with the 'ApiController' attribute indicates that a type and all derived types(controllers) are used to server HTTP API responses. Controller decorated with this attribute is configured with features and behavior targets at improving the developer experience for building APIs
* (Line: 7) Route to define our URL. The default expression '[Controller]' means name of the controller(eg: 'Cake'). We can define the custom name as the route that can be different from the controller name also.
* (Line: 8) Our C# class to become a controller it needs to inherit 'Microsoft.AspNetCore.Mvc.ControllerBase'.
* (Line: 10-14) Injected our DatabasContext into the controller.

**Read Operation:**

Let's create an Action method(which is a logical unit of the controller) of HTTP GET requests. Our job our action method to read the data from the database.

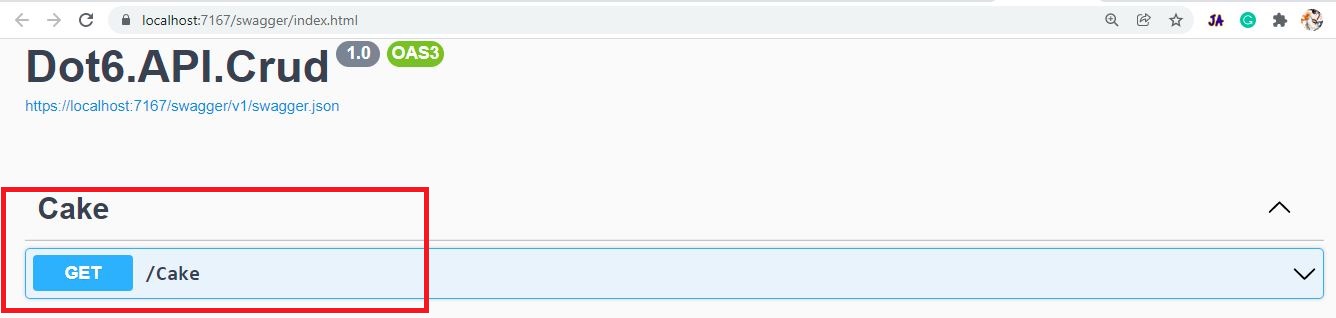
**Controllers/CakeController.cs:**

1. [HttpGet]
2. **public** **async** Task<IActionResult> GetAsync()
3. {
4. **var** cakes = **await** \_myWorldDbContext.Cake.ToListAsync();
5. **return** Ok(cakes);
6. }

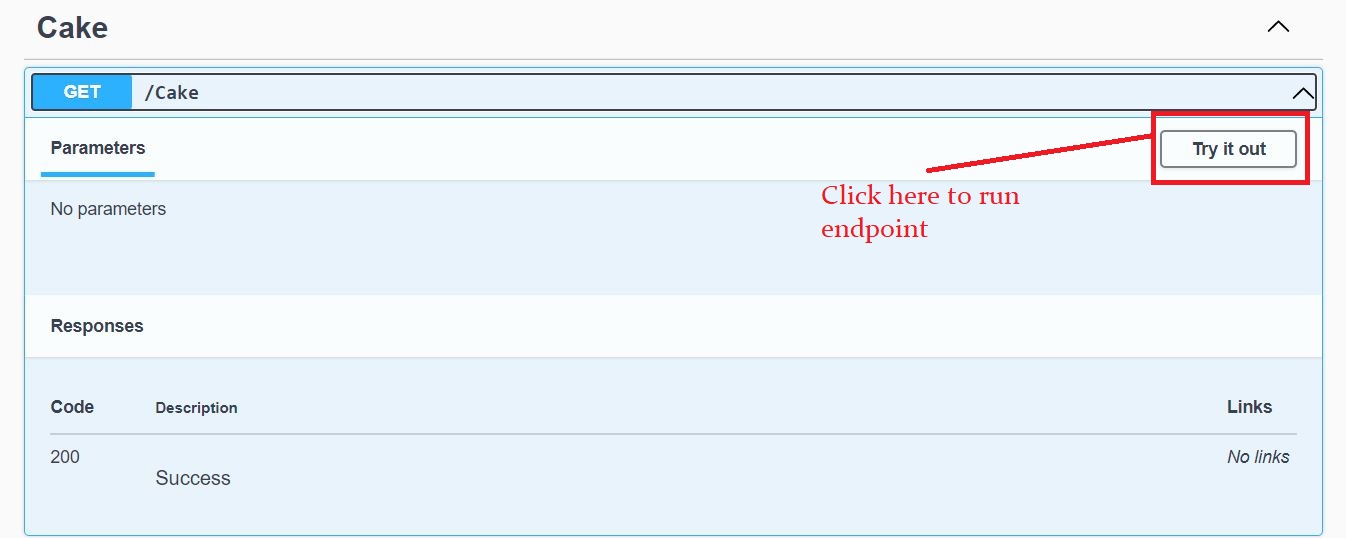
* Here we can observe there is no 'Route[]' attribute defined on the action method. If our controller contains a single HTTP Get action method it is optional to add the 'Route[]' attribute. By convention, API will return the response. But if we have multiple action methods of the same HTTP types like 'GET', 'POST', 'PUT', 'Delete' then we have to define routes per action method explicitly.
* (Line: 1) The 'HttpGet' attribute represents that our action method gets invoked for HTTP GET requests.
* (Line: 2-6) Defined an asynchronous action method to return the collection of data.
* (Line: 4) Fetching all records from the 'Cake' entity as an async call. The 'ToListAsync' is an extension from 'Microsoft.EntityFrameworkCore'.
* (Line: 5) The 'Ok()' method creates an OkObjectResult, object that produces an StatusCodes.Status200Ok(which means 200 success) response.

Now run the application and use the swagger page tool to test our newly create endpoint.

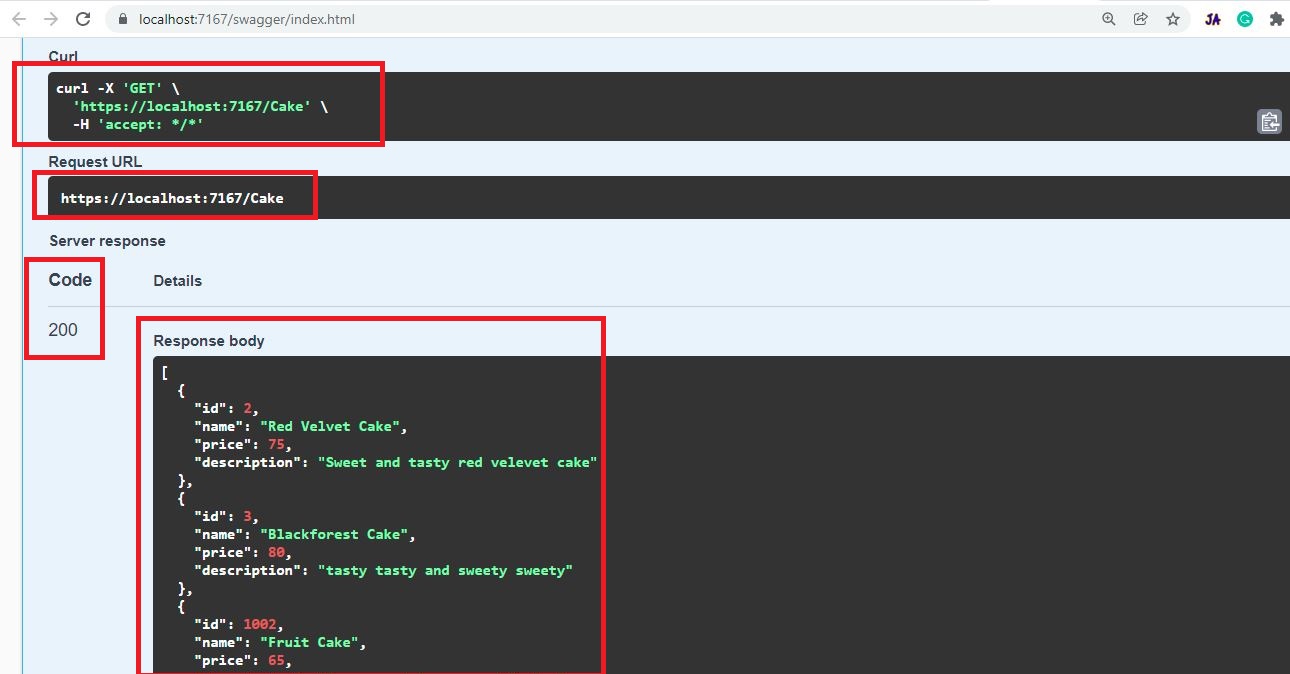
Step1:

[](https://blogger.googleusercontent.com/img/a/AVvXsEicuYEKE9eWNExkqMFo5Emt7Al1S1LBTHouE7J9GjZr1CcrO-4qn_efPLuiw3apRB4mBJXvpx44saG1q-S8rxj4WZs2c0RutUBoRi56Rb02pmo0A1eeWxzK4L9EIRXOz57vjvlqa_nt5yvZ0pmPGd-NCk_Ri-esHqvkeEWj14b9OGPOI8Gqturp0X5I=s1336)

Step 2:

[](https://blogger.googleusercontent.com/img/a/AVvXsEgHK2xGGtNbuuzc1aT_MfkSP9Xx9WJRxHX3Rwrhawp9luEC9Iu5HE995qye-zzeVBWwjHHTGVj9eLvj0-l3CvQqiY8mLcQrB7zgzTtbIMAQdfluRSQCQAUdF4iFoAJ5IHTLNLlA7r0tv1DVqG8u9I4j13HWHegMYejRRTX2SwCpv5EU9IP9R7DJWtUv=s1346)

Step3:

[](https://blogger.googleusercontent.com/img/a/AVvXsEhcyYo5ywoSuI5b5FEmaO_bcjg7799C0djrJG0csBFzcfc43R0lf0TT_2UI-0C4Ys5Ixs8SDnJbgqftAgTe8K56NKcGmNymmsrcfkUPTCAqHH3IwpVylQB-cxrlG8yRC-aWOe5_Br6xmgSvj9N03JaCgPJxNue_fTdylOBgAvKHHpRiHmrVpoB8MTm4=s1290)

**Create Operation:**

Let's create an action method for the HTTP POST request. Our action method job is to create a new record into the database.

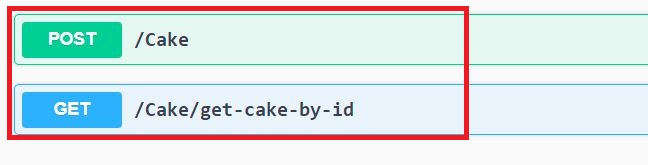
***Controllers/CakeController.cs:***

1. [HttpGet]
2. [Route("get-cake-by-id")]
3. **public** **async** Task<IActionResult> GetCakeByIdAsync(int id)
4. {
5. **var** cake = **await** \_myWorldDbContext.Cake.FindAsync(id);
6. **return** Ok(cake);
7. }
9. [HttpPost]
10. **public** **async** Task<IActionResult> PostAsync(Cake cake)
11. {
12. \_myWorldDbContext.Cake.Add(cake);
13. **await** \_myWorldDbContext.SaveChangesAsync();
14. **return** Created($"/get-cake-by-id?id={cake.Id}", cake);
15. }

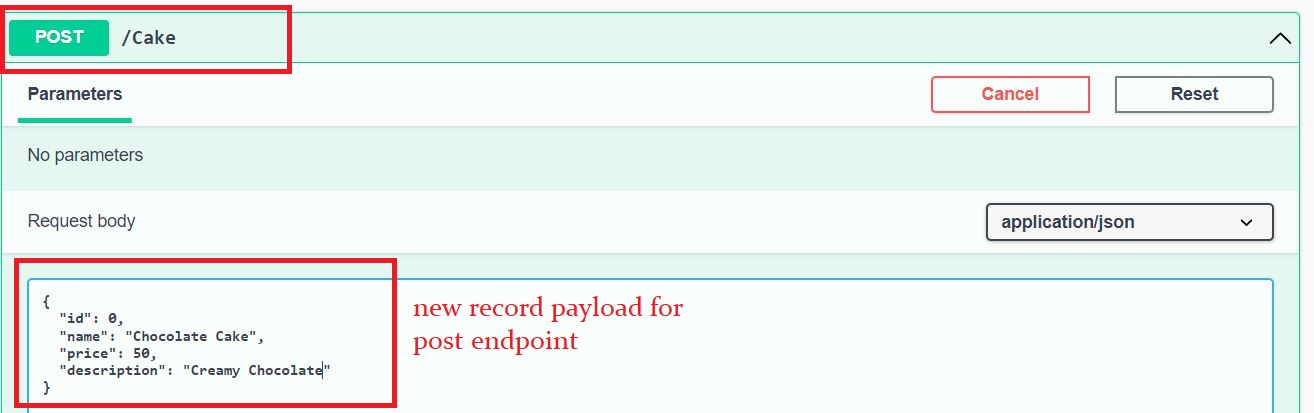
* (Line: 9) The 'HttpPost' attribute represents our action method only gets invoked for the HTTP POST requests.
* (Line: 10-15) Asynchronous action method for creating the new record.
* (Line: 12) The 'Add' method on database context begins tracking the given entity. This will update the state of the database context as 'EntityState.Added' which means data is ready to save into the database.
* (Line: 13) The 'SaveChangesAsync()' method saves our data into the database.
* (Line: 14) The 'Created()' method creates a CreatedResult object that produces a status code of 201(created) as a response. This 'Created()' method first parameter will be URL for endpoint where we can get the newly created record, so to generate this URL I have created another HTTP GET action method at(Line: 1-7) that fetches data by the 'id' value. If you want you can use the 'Ok()' method as a return type from the HTTP POST action method as well.

Now run the application and test our HTTP POST endpoint.

Step1:

[](https://blogger.googleusercontent.com/img/a/AVvXsEimJi_vYVoBjd8ywinOPXAWa5fxWl3IaIinDjJ79mK-jTA-cV0sEEU7OemFJ_twqGCTxMBU_n8VIvIO_6tupi1HR5vTi72DtVrwscaW6Dlhxhwcvt07rjrgrUprv3c-zi1L8AmF7ZkbU6VlsRM_HoBfz7ZqSWMWel6O_8I_6uiNygf3S1DXTIBy9_-8=s648)

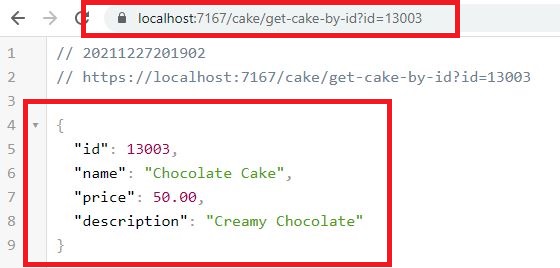
Step 2:

[](https://blogger.googleusercontent.com/img/a/AVvXsEi4B8m9e_6ektLeyj_m7HSf1e5-nzKI8d7hLtKI2-61w-IrIsCDEvsH1XyvbL8PUYbpOSpnSuMJmidFsQDPpungYXT30mx_68SKBN9BYH6KQ42dkMRTpbQqGJJWixqzmeZYmkB7bKcAyu19OvdlTg9bhGBNvQ3OHzYVUB2utXPWRNJTwz66WKLBfmuH=s1314)

Step3:

[](https://blogger.googleusercontent.com/img/a/AVvXsEgs238YTXg8B0TKy2QqxFTVu5uluqESYkMYWWEkS1_QRpVsubUoi2kCLsxZNr1f6nPz_VrlV0FUqcHdhaxb4mwDkGfFpdobxyl2QEH9rZChGN--9OpMmVS6t2jw_Ps7qc04Ocwkqbex1krJn8DYeAPe6SGPdnOSo2hETe-t-lcnWot7dXEpFOHbl_I5=s1314)

Step4:

[](https://blogger.googleusercontent.com/img/a/AVvXsEhALCYAUe8AkBAfmdyyMNG2GEkVFqTIaC14hstChz1MQkVw9piy_m6Y_ZTXb-wGzUmm0FtC4wIrcg72NY4VYSrKQP__bPQKhsLGB--hMkX-9NAs9eIs8tZjG82OMq4ZC9jkk9bizxZQb6i2TLbpHEU9kqqyoFp5PzbUCYSiz60cnDU_ozUhi2TkmNa3=s560)

**Update Operation:**

Let's create an action method for the HTTP PUT requests. Our action method job is to update the record.

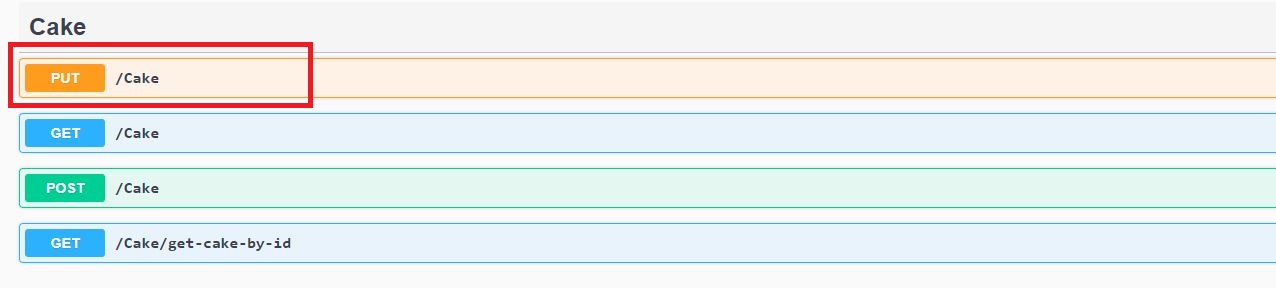
***Controllers/CakeController.cs:***

1. **[HttpPut]**
2. **public async Task<IActionResult> PutAsync(Cake cakeToUpdate)**
3. **{**
4. **\_myWorldDbContext.Cake.Update(cakeToUpdate);**
5. **await \_myWorldDbContext.SaveChangesAsync();**
6. **return NoContent();**
7. **}**

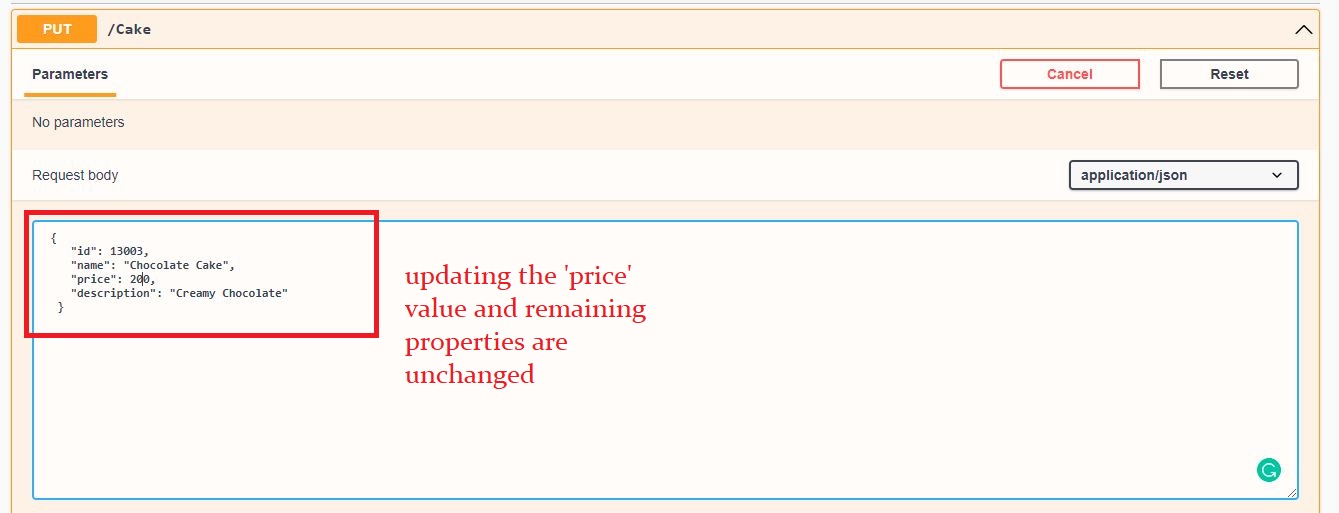
* (Line: 1) The 'HttpPut' represents our action method that gets invoked for HTTP PUT requests.
* (Line: 2-7) Asynchronous method for updating the records.
* (Line: 4) The 'Update()' method changes the Database context state as changed, which means data is ready to update. The 'Update()' method will update all columns for the record so make sure this is the case for the update or not. If you try to pass only a few columns of data then the remaining columns get updated with their default values. One more thing to remember the object should contain 'Id'(primary key) with value.
* (Line: 5) The 'SaveChangesAsyn()' method saves the data into the database.
* (Line: 6) The 'NoContent()' method creates a 'NoContentResult' object that produces an empty of status code 204(no content) as response.

Now run the application and test our HTTP PUT endpoint.

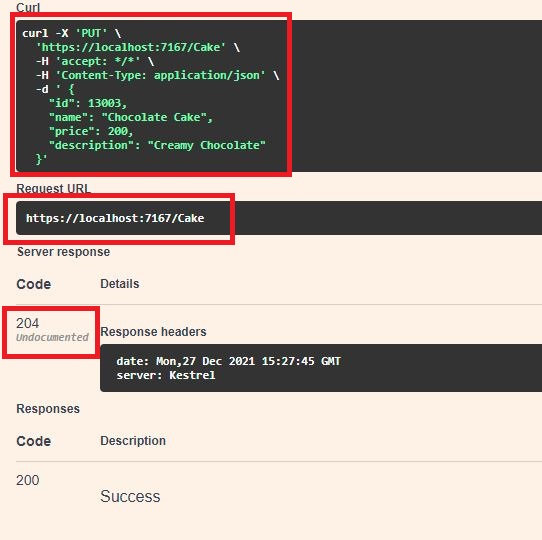
Step1:

[](https://blogger.googleusercontent.com/img/a/AVvXsEhCqL8sNhedOArXvSJ6F7w-WVqlp2jBD9KYmKzISdrjUGYH-j5f-SXurF1WScAEgTv9KC0klW7BNQNPSWK4K6aH_IhAsndyQ4jvPylG-WQSTFLWP8lHcR7xUv8mGO7rNQsOyAvk8VHIaGocpBeHdDszP8DfATTjRCSAKauGFkGtQPRROhcscoVYjfkB=s1276)

Step2:

[](https://blogger.googleusercontent.com/img/a/AVvXsEhTOj4qZW5iT1whv2MucxjdqfjQRyKjVXK4mGQEixZVccPRctaiBph0W6Jp1_nGNUSGofAck_a_eZ2scuO_lLZRYslAJ6bJNrSYQEE3iZ5B0non05uTMIk4Cfl1e0GPA0KPGyaLhnJOIUPdEOb0DOpZnhdX92kulMQUy6a__EIbcPiJxeNsP_GXddyY=s1339)

Step3:

[](https://blogger.googleusercontent.com/img/a/AVvXsEjCShqXTW3DQ-FHisw_ME5N26-qjddjFYtIYl5z7RY0QtUiFjSvZK0sB0fpNTGgiVpUVFD9FYzs3n4lEGJfBptunSYzbsFP2tgR1aKBvsL22ZijyZwLQqNDwlKzJltZnxEzwiQ-OCQ3ToJVQQ1ppHruZncddSUihluZjxazzCqbD47QpIgYMYtK9L3P=s542)

Step4:

[](https://blogger.googleusercontent.com/img/a/AVvXsEhxaCVZq805RCzZxgM_7fMdPkNNGkbB-xFm9itqOAqtGuahyKXeEFOgXT91MRubUXt9Uc47sacBGvAY-8JWDe1ZMyI8yvVoBMC_XVDRbndI-dsP7s3rF34fsLJ4GJP1QlrTD_yA_TF4gVSVgEvs08FdmpV66ZWN176sOtDp1fg-b5E0gnxMJ9Y3nXbx=s504)

**Delete Operation:**

Let's create an action method of HTTP Delete request. Our action method job is to delete a particular(eg: by 'id' value) record from the database.

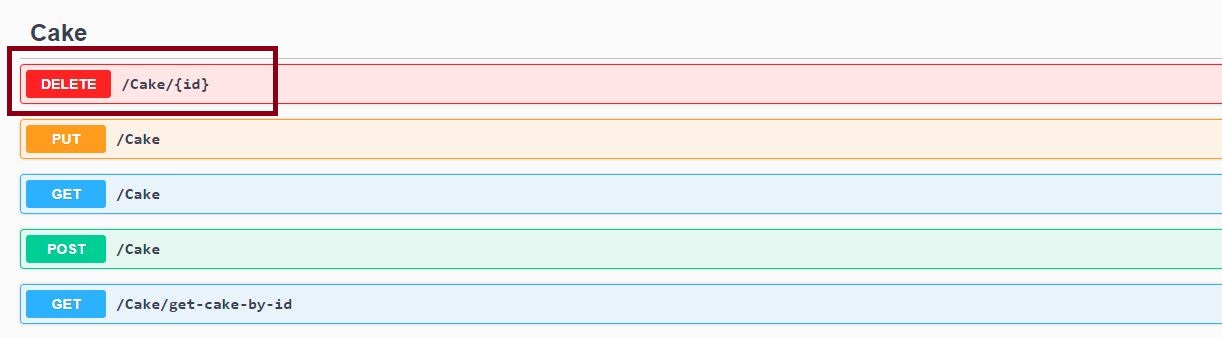
***Controllers/CakeController.cs:***

1. [Route("{id}")]
2. [HttpDelete]
3. **public** **async** Task<IActionResult> DeleteAsync(int id)
4. {
5. **var** cakeToDelete = **await** \_myWorldDbContext.Cake.FindAsync(id);
6. **if** (cakeToDelete == **null**)
7. {
8. **return** NotFound();
9. }
10. \_myWorldDbContext.Cake.Remove(cakeToDelete);
11. **await** \_myWorldDbContext.SaveChangesAsync();
12. **return** NoContent();
13. }

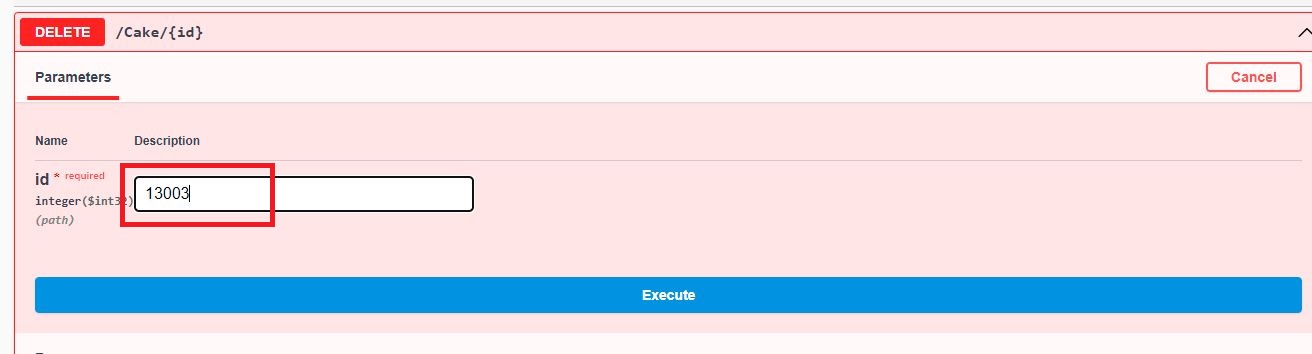
* (Line: 1) The '[Route]' attribute defined that means 'id' value should be passed as part of URL, and the '{}' is expression means any string defined inside of it will be the input parameter to the action method.
* (Line: 2) The 'HttpDelete' attribute represents our action method getting invoked only for the HTTP Delete methods.
* (Line: 3-13) Asynchronous action method for deleting the record from the database.
* (Line: 5) Fetching the record that needs to be deleted from the database.
* (Line: 6-9) Validating the record to be deleted. If no valid record in the database then we return the 'NotFound()' method whose status code is '404'.
* (Line: 10-11) Removing the record from the database context. The 'SaveChangeAsync()' update the database context changes to the database.
* (Line: 12) The 'NoContent()' status code is '204'.

Now run the application and test our HTTP DELETE endpoint.

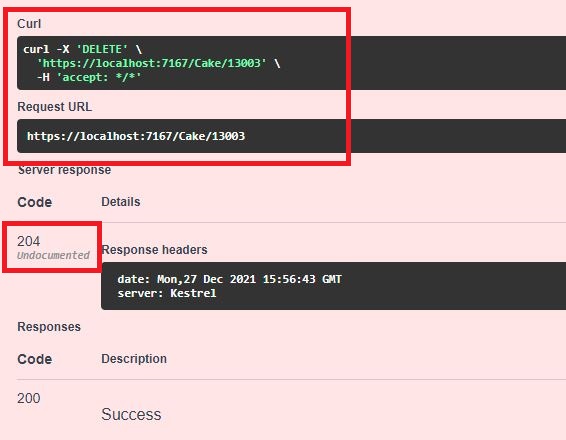
Step1:

[](https://blogger.googleusercontent.com/img/a/AVvXsEjWlwcfwzmNB9oRFkJmJ69g9ALyz2UJ7d2McZcerAAhoTUcCumqB0RWPnhkk7QojTnkxhdQ-hRjuNJrFnbmxZtXG2ytPkwUQJ2I7dyA6Ve0JqtI64LMwPMrMFZYG95dY3syeMBpiTgZgBym7V5zyUTf4kW9MBgfmJ-bxoaqX2Uy9KLNpMMDnY2AQEAf=s1222)

Step:2

[](https://blogger.googleusercontent.com/img/a/AVvXsEhWNbXPBsNNsGLux8ZnHudbYbzMdVtRHcTgXi0MgWwBRZ1kNyM7CjxK5coRwFFlP8KzUX6ZErw03TgB_XdaUdFUZy36dpA9d48_JmsbCTbnzhl8dg6rdxCerxq4XGbAbDjDRiXW8Rgf_mxtOCZlU9jr9fwT1xVTJeHph-vuvAUs3_gKoiHdHIhPVWLy=s1312)

Step3:

[](https://blogger.googleusercontent.com/img/a/AVvXsEjb73ooOZJpzr5woLKKP3RZgS5z7tABBjkmQFaPzDTLlZ8L55oW34lBeGad99_6goVk7XweBW4SopZqI57too5H2UlSiepjbgsh5PWSDQ0g48knET78o16Bym7TvHaReI9kW1hLfX-YU_u1rw-iks4HS2Tx_fKghEFc-JYl5xizQTNP5rvNxro4yjiy=s566)