* **API** is an application programming interface. It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it
* An Application Programming Interface (API) allows two systems to communicate with one another
* APIs can use HTTP requests to get information from a web application or web server.

**What are the parameters in API?**

<https://www.soapui.org/rest-testing/understanding-rest-parameters.html>

There are four types of parameters: header parameters, path parameters, query string parameters, and request body parameters

Parameters are options you can pass with the endpoint (such as specifying the response format or the amount returned) to influence the response

* [**Header parameters**](https://idratherbewriting.com/learnapidoc/docapis_doc_parameters.html#header_parameters): Parameters included in the request header, usually related to authorization.
* [**Path parameters**](https://idratherbewriting.com/learnapidoc/docapis_doc_parameters.html#path_parameters): Parameters within the path of the endpoint, before the query string (?). These are usually set off within curly braces.
* [**Query string parameters**](https://idratherbewriting.com/learnapidoc/docapis_doc_parameters.html#query_string_parameters): Parameters in the query string of the endpoint, after the ?.
* [**Request body parameters**](https://idratherbewriting.com/learnapidoc/docapis_doc_parameters.html#request_body_parameters): Parameters included in the request body. Usually submitted as JSON.

**Call API from C#**

<https://www.aspsnippets.com/Articles/Read-response-from-Web-API-using-HttpClient-in-C.aspx>

static void Main(string[] args)

{

    Start:

    Console.Write("Enter Name: ");

    string name = Console.ReadLine();

    string apiUrl = "http://localhost:26404/api/CustomerAPI";

    var input = new

    {

        Name = name,

    };

    string inputJson = (new JavaScriptSerializer()).Serialize(input);

    HttpClient client = new HttpClient();

    HttpContent inputContent = newStringContent(inputJson, Encoding.UTF8, "application/json");

    HttpResponseMessage response = client.PostAsync(apiUrl + "/GetCustomers", inputContent).Result;

    if (response.IsSuccessStatusCode)

    {

        List<Customer> customers = (new JavaScriptSerializer()).Deserialize<List<Customer>>(response.Content.ReadAsStringAsync().Result);

        if (customers.Count > 0)

        {

            foreach (Customer customer in customers)

            {

                Console.WriteLine(customer.ContactName);

            }

        }

        else

        {

            Console.WriteLine("No records found.");

        }

    }

    Console.WriteLine();

    goto Start;

}

public class Customer

{

    public string CustomerID { get; set; }

    public string ContactName { get; set; }

    public string City { get; set; }

}

**Asynchronous**

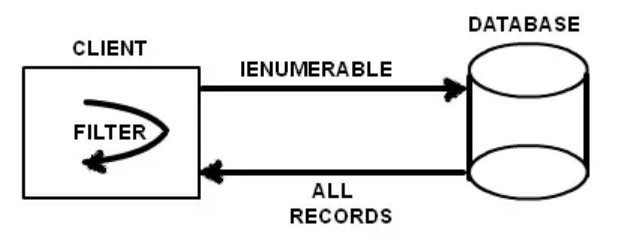
* When we are dealing with UI and on button click, we use a long running method like reading a large file or something else, which will take a long time, in that case, the entire application, must wait to complete the whole task
* In other words, if any process is blocked in a synchronous application, the entire application gets blocked and our application stops responding until the whole task completes
* Asynchronous programming is very helpful in this condition. By using Asynchronous programming, the Application can continue with the other work that does not depend on the completion of the whole task
* We will get all the benefits of traditional Asynchronous programming with much less effort by the help of async and await keywords
* Suppose, we are using two methods as Method1 and Method2 respectively and both the methods are not dependent on each other and Method1 is taking a long time to complete its task. In Synchronous programming, it will execute the first Method1 and it will wait for completion of this method and then it will execute Method2. Thus, it will be a time intensive process even though both the methods are not depending on each other
* The beginning of an async method is executed just like any other method. That is, it runs synchronously until it hits an “await” (or throws an exception).

**Endpoint**

* API endpoint is the point of entry or point of contact in a communication channel when two systems are interacting
* endpoint is the place of interaction between applications
* API endpoints are the specific digital location where requests for information are sent by one program and where the resource lives

**Ienumerable & IQueryable**

* Both are interfaces
* IQueryable inherits from Ienumerable
* Means, whatever things ienumerable can do, IQueryable can also do
* But some things iQueryable can do which is ienumerable can’t do
* The main difference between “IEnumerable” and “IQueryable” is about where the filter logic is executed. One executes on the client side (in memory) and the other executes on the database.
* For example, we have 10000 records for User in our database and let say only 900 out which are active users, so in this case if use “IEnumerable” approach then first it load all 10000 records in the memory then apply the IsActive filter on it which eventually returns the 900 active users.(Two steps process and load all the records in memory as well which will make the process slow)  
  While on the other hand on the same case if we use “IQueryable” approach it will directly apply the IsActive filter on the database which directly from there will return the 900 active users.(One step process and will return less records which will be much faster).
* IEnumerable<T>:



* IQueryable<T>

