# What is Api?

Api Stands for **Application Programming Interface** through which software can interact with each other.

VS Studio🡪New Project🡪ASP.NET Web Application (.NET Framework)🡪Create🡪Empty🡪Uncheck HTTPS🡪Check Web API

In App\_Start 🡪WebApiConfig.cs we define the route (basically the url)🡪

* In Asp.net MVC we don't have a keyword api in the url.

someurl.com/Action

* In Asp.net Web Api we have a keyword api in the url.

someurl.com/api/Action

* In Rest Api's we don't use the term CRUD (Create, Read, Update, Delete)
* We use HTTP Verbs.
  + - Get 🡪 Read
    - Post 🡪 Create
    - Put 🡪 Update
    - Delete 🡪 Delete

To retrieve a data in web API, we need to use get () method.

# Code First Approach

* We don't design database
* We don't create Tables
* Entity Framework is responsible for all kind of things

Manage NuGet🡪



Connection string 🡪

<https://learn.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/creating-a-connection-string>

<connectionStrings>

<add name="QuotesDBContext" connectionString="Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=aspnet-QuotesDb;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|\QuotesDb.mdf" providerName="System.Data.SqlClient" />

</connectionStrings>

Get Method 🡪

QuotesDbContext \_quotesDbContext = new QuotesDbContext();

// url: GET: api/Quotes

public IEnumerable<Quote> Get()

{

//This Get method will return all the contents from the DB

return \_quotesDbContext.Quotes;

}

// url: GET: api/Quotes/5

public Quote Get(int id)

{

var quote= \_quotesDbContext.Quotes.Find(id);

//Find(id) method will pick the specific quote from the database against the Id

return quote;

}

Post Method 🡪

// POST: api/Quotes

public void Post([FromBody] Quote quote)

{

\_quotesDbContext.Quotes.Add(quote); ////Quotes is the table Name

\_quotesDbContext.SaveChanges(); //// SaveChanges method of ObjectContext is a gateway to persist all changes made to entities to the database.

}

Put Method 🡪

// PUT: api/QuotesTwo/5

public void Put(int id, [FromBody] Quote quote)

{

var entity = \_quotesDbContext.Quotes.FirstOrDefault(q => quote.Id==id);

entity.Title = quote.Title;

entity.Author = quote.Author;

entity.Description = quote.Description;

\_quotesDbContext.SaveChanges();

}

Delete Method 🡪

// DELETE: api/QuotesTwo/5

public void Delete(int id)

{

var quote =\_quotesDbContext.Quotes.Find(id);

\_quotesDbContext.Quotes.Remove(quote);

\_quotesDbContext.SaveChanges();

}

# What is http status code ?

Http status codes are standard response codes given by website servers on the internet. The codes help identify the cause of the problem when a web page or other resource doesn't load properly. Http status codes are sometimes called **browser error codes or internet error** codes.

Whenever we want to add status code, we have to face some kind of error and that is we cannot return the status code with this method because this method is of type **IEnumerable**. So, in order to return the status code, we will need to set the type of function to **IHttpActionResult.**

public IHttpActionResult Get()

{

//This Get method will return all the contents from the DB

var quote = \_quotesDbContext.Quotes;

return Ok(quote);

}

public IHttpActionResult Get(int id)

{

var quote= \_quotesDbContext.Quotes.Find(id);

//Find(id) method will pick the specific quote from the database against the Id

if (quote == null)

{

return NotFound();

}

else

return Ok(quote);

}

If you want to implement the success status code or 200, then simply call the **Ok**() method and just pass the code object. And this method will return the status code 200 as well as the list of codes. If you want to return the 400 bad request error, then simply return the **BadRequest**() function. If you want to return 404 not found error, then simply return the **NotFound**() method.

But what if you want to return some other status code? The shortcut method doesn’t cover all the possible status codes. For this purpose, we will use the StatusCode function which accepts HttpStatusCode.

let's try to add this status code in the post method.

public IHttpActionResult Post([FromBody] Quote quote)

{

\_quotesDbContext.Quotes.Add(quote);

\_quotesDbContext.SaveChanges();

return StatusCode(HttpStatusCode.Created);

}

let's start with the put request.

public IHttpActionResult Put(int id, [FromBody] Quote quote)

{

var entity = \_quotesDbContext.Quotes.FirstOrDefault(q => quote.Id == id);

if (entity == null)

{

// return StatusCode(HttpStatusCode.BadRequest);

//or

return BadRequest("ID not found");

}

else {

entity.Title = quote.Title;

entity.Author = quote.Author;

entity.Description = quote.Description;

\_quotesDbContext.SaveChanges();

// return StatusCode(HttpStatusCode.Accepted);

//or

return Ok("ID Updated!");

}

}

let's apply the status code on the delete request.

public IHttpActionResult Delete(int id)

{

var quote =\_quotesDbContext.Quotes.Find(id);

if (quote == null)

{

// return StatusCode(HttpStatusCode.BadRequest);

//or

return BadRequest("ID not found");

}

else

{

\_quotesDbContext.Quotes.Remove(quote);

\_quotesDbContext.SaveChanges();

//return StatusCode(HttpStatusCode.OK);

//or

return Ok("ID Deleted!");

}

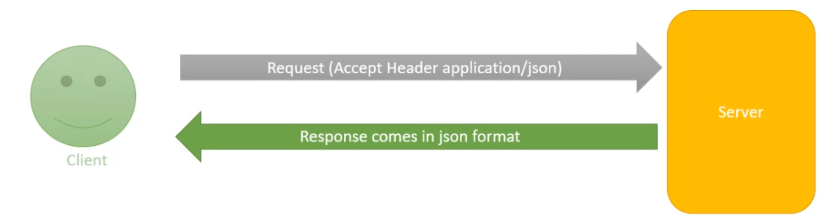
}

# What is Content Negotiation ?

One of the standards of the restful services is that the client should be able to decide the format of response from the server. For example, if the client wants the response in XML or in Json format and so on.

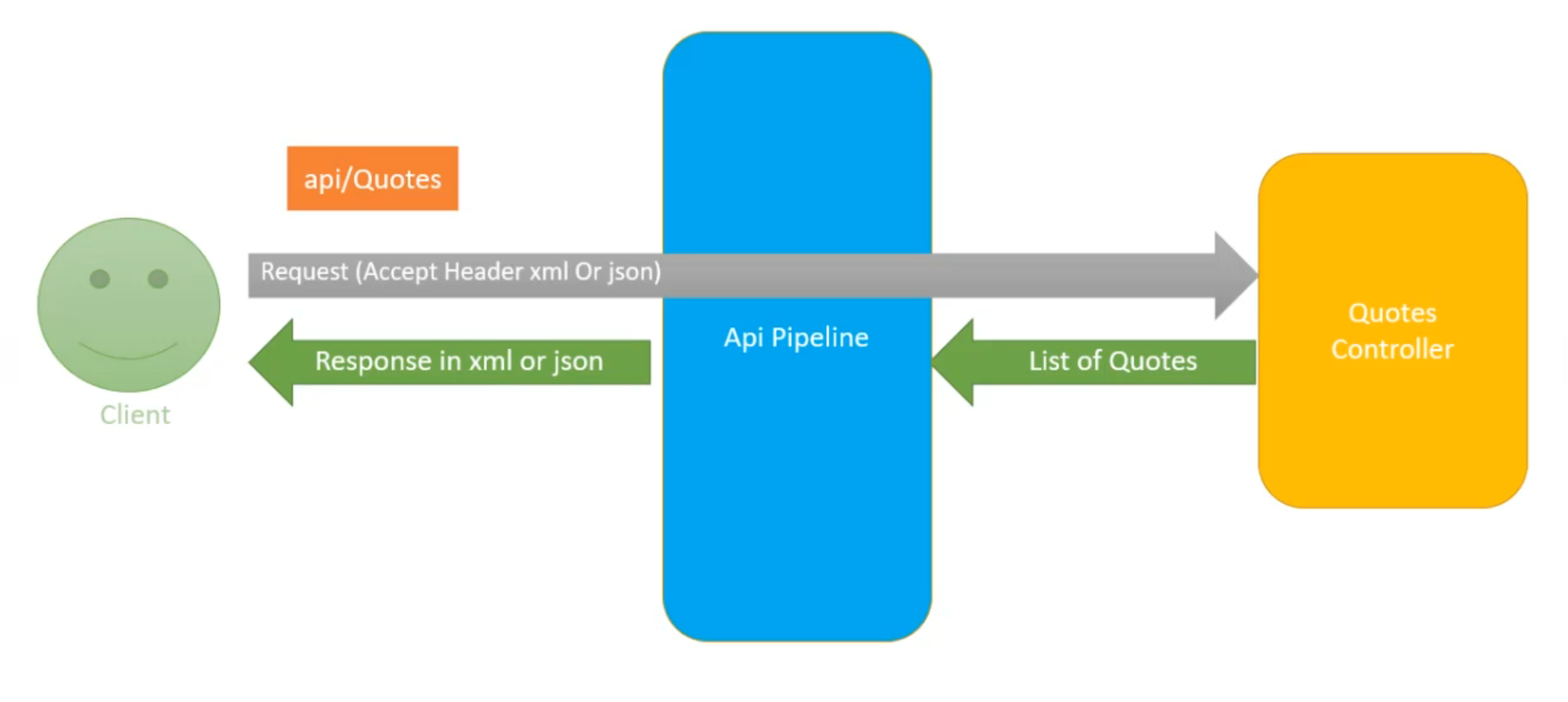


When a client sends a request to the server, the request includes something called **accept header**. Using this **accept header**, the client can specify the format which they want from the server.



For example, if the accept header is XML. The server sends the format in XML. If it is Json, then the server sends the format in Json format.

**The default response will be in the Json format...**



The web API generates the controller data. That we want to send to the client. In our example, the QuotesController, we are generating the Quotes list which we want to return to the client once the list of the Quotes is generated. The job of the controller is done.

It's going to hand that list to the web API pipeline, which then looks at the **accept header** and depending upon the format which client has requested, Web API will choose the appropriate formatter for formatting the data.

For example, if the client has requested for XML data, Web API uses XML formatter if the client has requested for Json data. Web API uses Json formatter.

These formatters are nothing but classes and they are called **media type formatters**.

Now there is one interesting thing here, and that is if we will send the get request via **web browser**, then by **default the response will come in XML format**.

# Media Type Formatters

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

**config.Formatters.Remove(config.Formatters.XmlFormatter);**

}

Including the above command in WebApiConfig.cs will provide the accept header always in JSON format and never in xml format as we have removed the XmlFormatter. In browser also we will get in JSON format.

**config.Formatters.Remove(config.Formatters.JsonFormatter);**

Including the above command in WebApiConfig.cs will provide the accept header always in XML format and never in JSON format as we have removed the JsonFormatter. In browser also we will get in XML format. Now let

* If a client sends a request from the web browser, then he'll get the Json data.
* If a client sends a request from the postman, then it's up to the client in which format he wants the data.

**config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/html"));**

The new **MediaTypeHeaderValue** that comes from the namespace using System.Net.HTTP.Headers. So, when the media type format is **text/html**, we want to use the Json formatter to format the data because by default the formatter in web browser is **text/html.**

config.Formatters.Remove(config.Formatters.XmlFormatter);

config.Formatters.JsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;

These two lines will responsible for indented JSON formatting in browser.

# Code First Migrations

After adding some extra property in the model class, go to Package manager console🡪command🡪enable-migrations

Then in solution explorer a new folder solution will come.

console🡪command🡪add-migration <name>

console🡪command🡪update-database

# Model Validation

So, in order to add the validation, we will need the **data annotation**. So, let's understand the data annotation with an example. Data annotation in .Net framework means add extra meaning to the data by adding attribute tags. The advantage of using data annotation feature is that by applying the data attributes, we can manage the data definition in a single place and don't need to rewrite the same rules in multiple places.

using System;

using System.Collections.Generic;

**using System.ComponentModel.DataAnnotations;**

using System.Linq;

using System.Web;

namespace QuotesThreeAPI.Models

{

public class Quote

{

public int Id { get; set; }

[Required]

[StringLength(20)]

public string Title { get; set; }

[Required]

[MinLength(20)]

public string Author { get; set; }

[Required]

[MaxLength(200)]

public string Description { get; set; }

[Required]

public string type { get; set; }

[Required]

public DateTime CreatedAt { get; set; }

}

}

console🡪command🡪add-migration <name>

console🡪command🡪update-database

In controller

public IHttpActionResult Post([FromBody] Quote quote)

{

**if(!ModelState.IsValid)**

**{**

**return BadRequest(ModelState);**

**}**

\_quotesDbContext.Quotes.Add(quote);

\_quotesDbContext.SaveChanges();

return StatusCode(HttpStatusCode.Created);

}

public IHttpActionResult Put(int id, [FromBody] Quote quote)

{

**if (!ModelState.IsValid)**

**{**

**return BadRequest(ModelState);**

**}**

var entity = \_quotesDbContext.Quotes.FirstOrDefault(q => quote.Id == id);

if (entity == null)

{

// return StatusCode(HttpStatusCode.BadRequest);

//or

return BadRequest("ID not found");

}

else {

entity.Title = quote.Title;

entity.Author = quote.Author;

entity.Description = quote.Description;

\_quotesDbContext.SaveChanges();

// return StatusCode(HttpStatusCode.Accepted);

//or

return Ok("ID Updated!");

}

}