Build your own RESTful API from scratch:

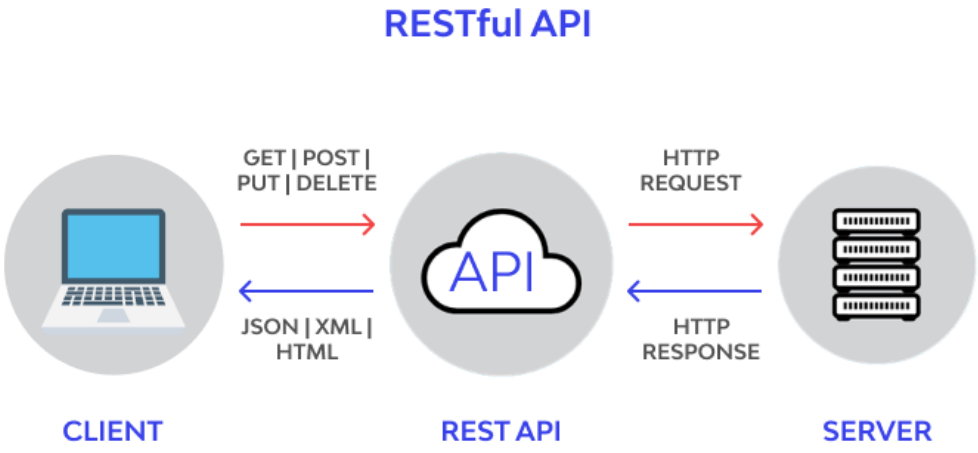
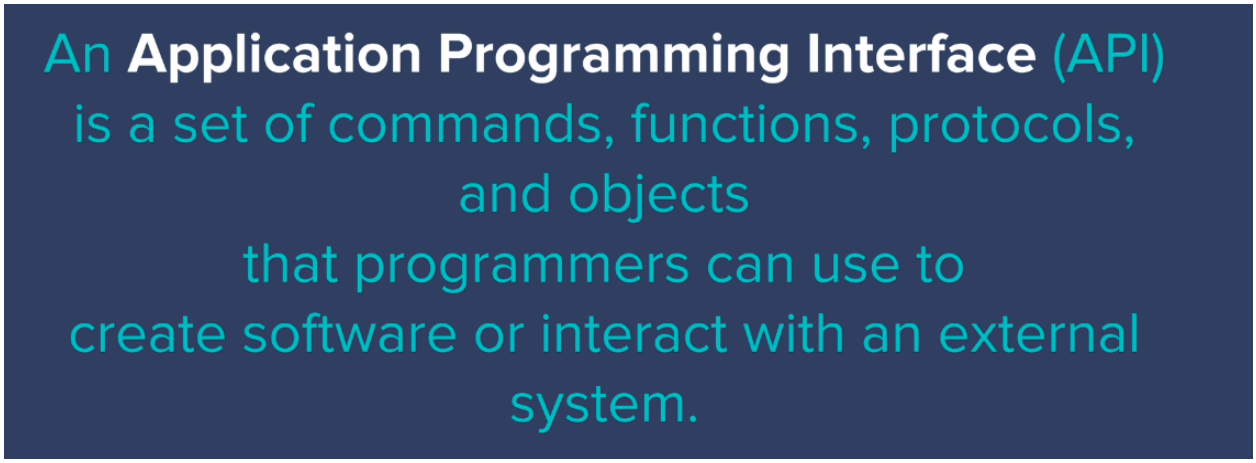


Table of Contents :

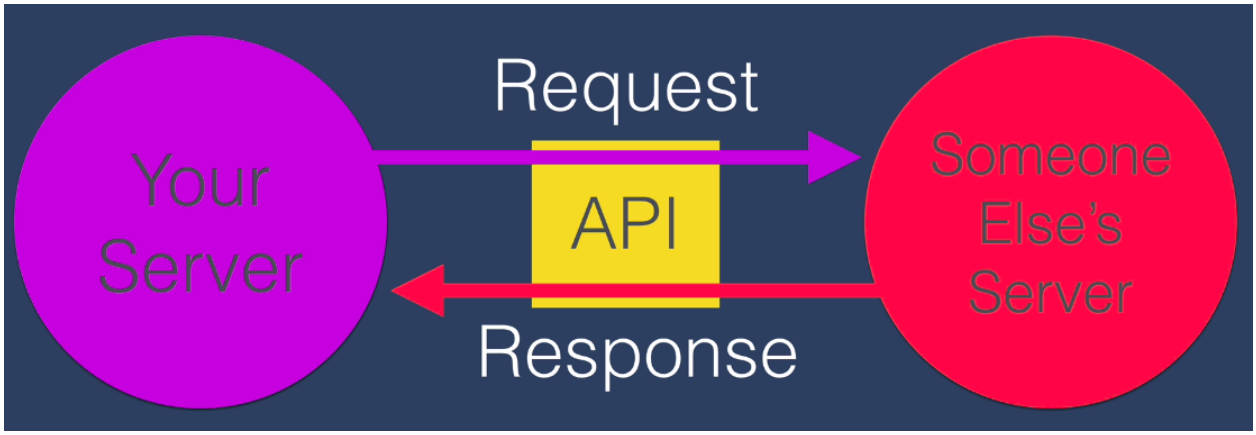
Introduction :

Since the invention of the internet, we have been using different applications and web pages to get data for various resources. However, where does this data come from?
Well, it's the servers from where we get the data. So in this article we will look into how a client communicates with the servers to extract the required information.

What is an API ?



APIs work using 'requests' and 'responses.'



API Endpoint :

- When an API requests information from a web application or web server, it will receive a response. The place that APIs send requests and where the resource lives, is called an **endpoint**.
- An endpoint is one end of a communication channel.
- For APIs, an endpoint can include a URL of a server or service. Each endpoint is the location from which APIs can access the resources they need to carry out their function.
- Twitter API Endpoint example : https://api.twitter.com/2/tweets/search/stream

API Path :

- An API URL Path is an address that allows us to access an API and its various features.
- There are 2 parts to any API URL:
  a) **Base URL :** the base address for the specific API that we are using. Until we choose a specific Endpoint, though, the Base URL isn't going to do much.
  e.g ***https://www.google.com*** acts as the Base URL.

  b) **Endpoint :** a specific "point of entry" in an API. We attach these to the end of our Base URL and get results depending on which Endpoint we choose.
  e.g ***https://www.google.com/support*** acts as the Endpoint.

API Parameter :

- API Parameters are options that can be passed with the endpoint to influence the response.
- In GET requests, they're found in strings at the end of the API URL path.
- In POST requests, they're found in the POST body.
- In the world of APIs, these are like search filters. And depending on the parameters we set, we get a different response each time.

e.g ***www.yoursite.com?myparam1=123&myparam2=abc&myparam2=xyz***

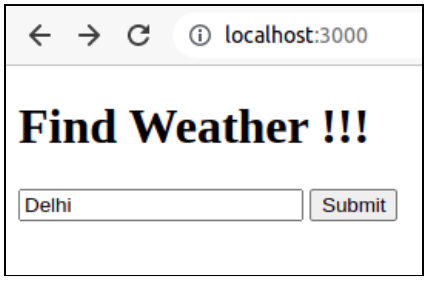
Well, the stuff at the end, after the .com :
?myparam1=123&myparam2=abc&myparam2=xyz are called parameters.
They are separated by &.

Using Weather Map API to fetch the temperature :

```
const express = require("express");
const https=require("https");
const bodyParser = require("body-parser");

const app=express();
app.use(bodyParser.urlencoded({extended:true}));

app.get("/",function(request,response){
    response.sendFile(__dirname+"/index.html");
});

app.post("/",function(request,response){

    var place=request.body.place;

    const
```

```
url="https://api.openweathermap.org/data/2.5/weather?q="+place+"&appid=9c32
34d1801c308ec2d2aca43ef657e3&units=metric";
    https.get(url,function(res)
    {
        res.on("data",function(data)
        {
            const weatherData=JSON.parse(data);
            const temperature=weatherData.main.temp;
            const weatherDesc=weatherData.weather[0].description;
            const icon=weatherData.weather[0].icon;
            const
imageUrl="http://openweathermap.org/img/wn/"+icon+"@2x.png";

            response.write("<p>The weather is currently
"+weatherDesc+"</p>")
            response.write("<h1>Temperature of "+place "= "+temperature"
degree Celsius</h1>");
            response.write("<img src="+imageUrl+">");
            response.send();
        })
    })

});

app.listen(3000,function(){
    console.log("Server started on port 3000");
});
```

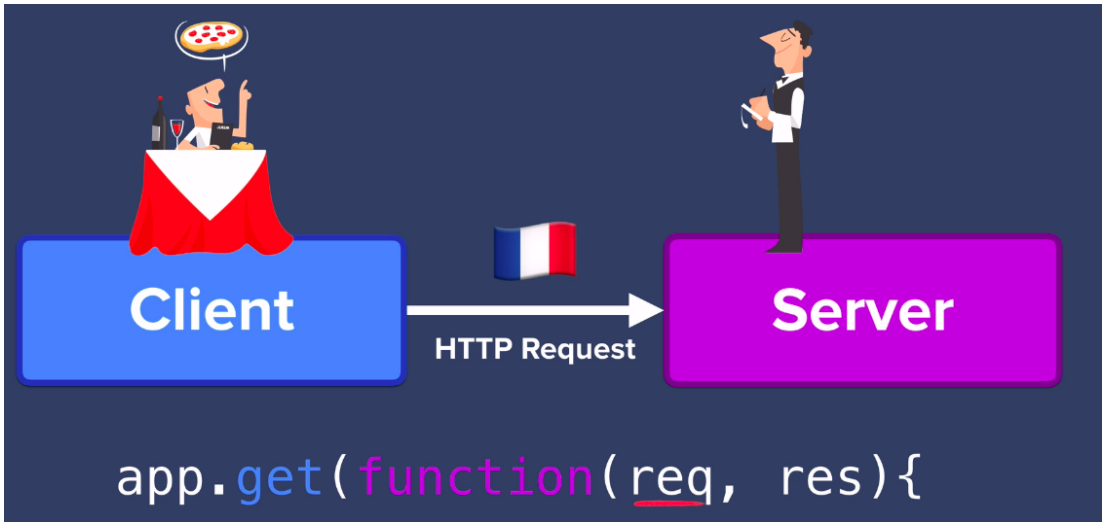
Input the Place:



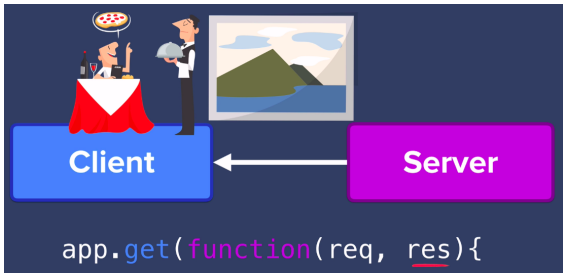
Output:

What is the REST API ?

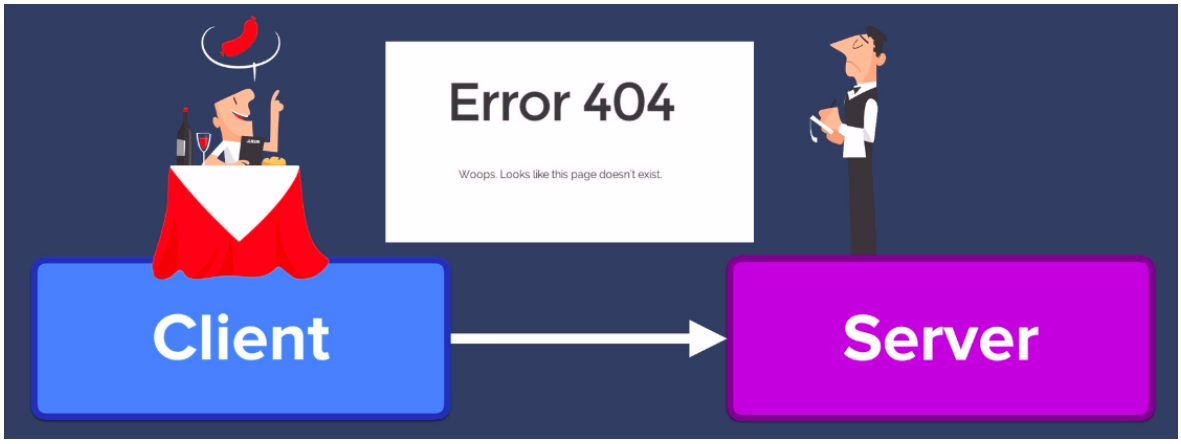
- The term REST stands for REpresentational State Transfer.
- It is an architectural style that defines a set of rules in order to create Web Services.
- In a client-server communication, REST suggests creating an object of the data requested by the client and send the values of the object in response to the user.



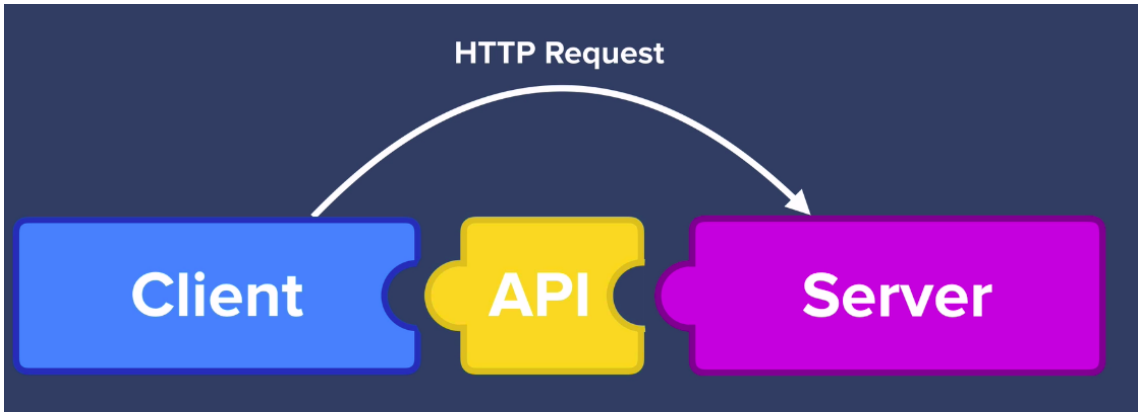
Client sends requests to the server using the .get function. It takes a function with request and response.
Server sends the response back : either HTML pages, or the images or data.



If the requested data is not found then the server will respond to Error 404 (Not found).

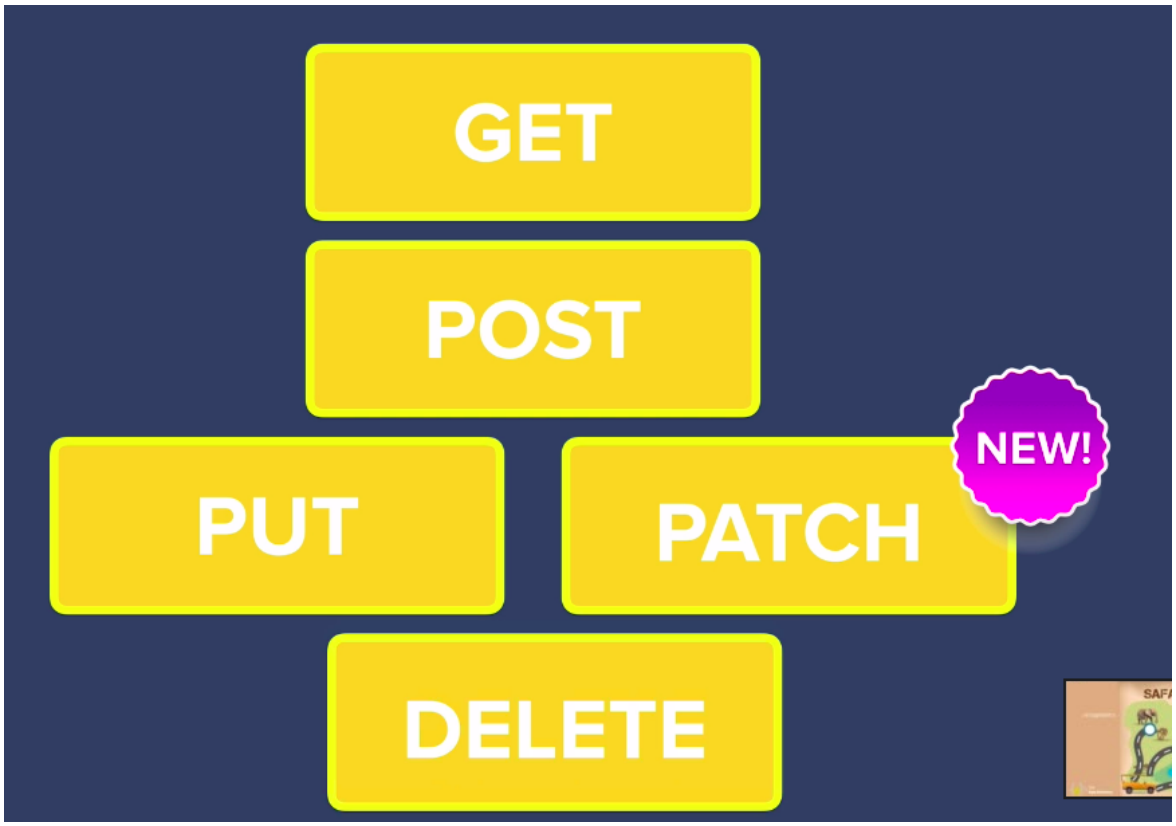


Now, the client interacts with the server using API's .



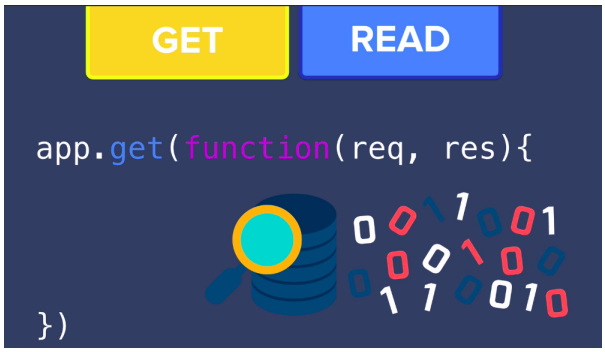
Rules to make an API → RESTful

To make an API restful, we need to use following **HTTP request verbs**:

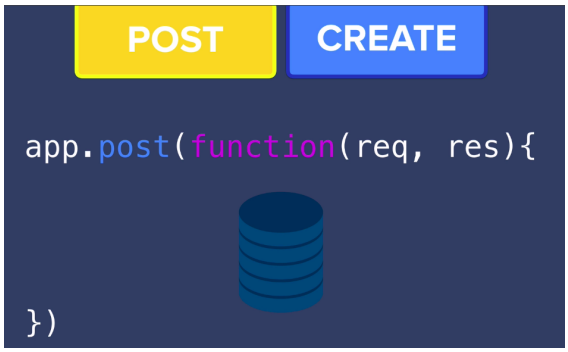


CRUD Operations:

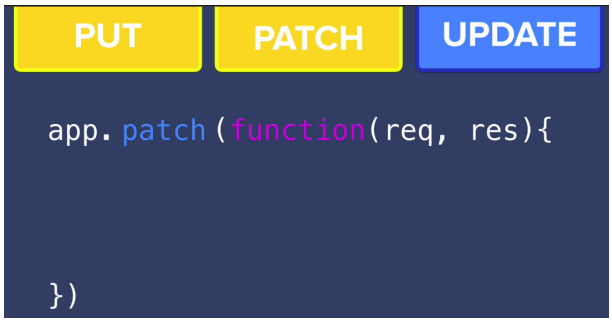
GET is equivalent to **READ**



POST is equivalent to **CREATE**

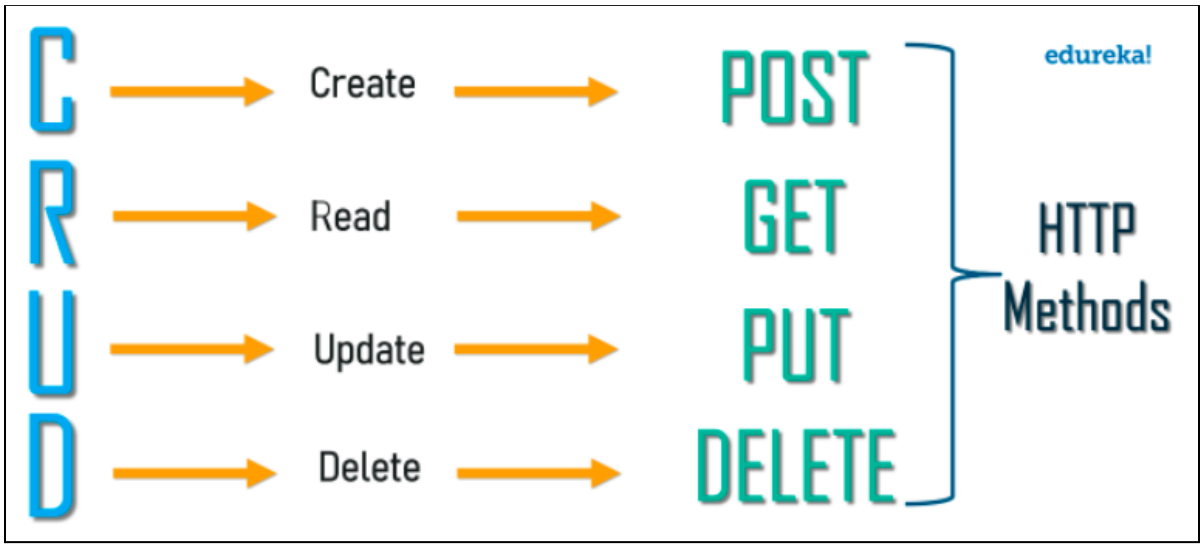


PUT, PATCH is equivalent to **UPDATE**



DELETE is equivalent to **DELETE**

Summary:



RESTful Routing :

RESTful Routing

HTTP Verbs	/articles	/articles/jack-bauer
GET	Fetches **all** the articles	Fetches **the** article on jack-bauer
POST	Creates **one** new article	-
PUT	-	Updates **the** article on jack-bauer
PATCH	-	Updates **the** article on jack-bauer
DELETE	Deletes **all** the articles	Deletes **the** article on jack-bauer

Difference between REST and SOAP:

REST	SOAP
• REST stands for REpresentational State Transfer. • REST is an architectural style that doesn't follow any strict standard. • REST is not restricted to XML and its the choice of implementer which Media-Type to use like XML, JSON, Plain-text. • REST has SSL and HTTPS. • REST can use SOAP.	• SOAP stands for Simple Object Access Protocol. • Since SOAP is a protocol, it follows a strict standard to allow communication between the client and the server. • SOAP uses only XML for exchanging information in its message format. • SOAP has SSL(Secure Socket Layer) and WS-security. • SOAP cannot use REST.

Creating a database with Robo3T

```
cd /usr/local/bin/robo3t/bin/
./robo3t
```


```
cd /usr/local/bin/robo3t/bin/
./robo3t
```

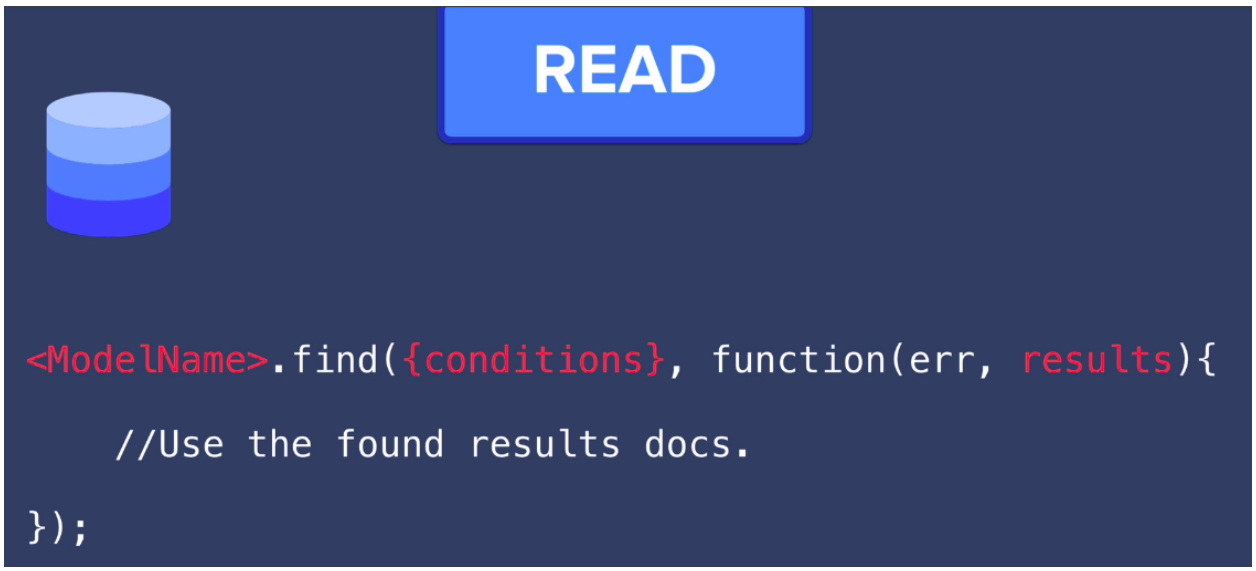
Server setup

Solution:

```
const express = require("express");
const bodyParser = require("body-parser");
const ejs = require("ejs");
const mongoose = require('mongoose');

const app = express();

app.set('view engine', 'ejs');

app.use(bodyParser.urlencoded({
  extended: true
}));
app.use(express.static("public"));

//TODO
mongoose.connect("mongodb://localhost:27017/wikiDB");
const articleSchema={
  title: String,
  content: String
}

const Article=mongoose.model("Article",articleSchema);

app.listen(3000, function() {
  console.log("Server started on port 3000");
});
```

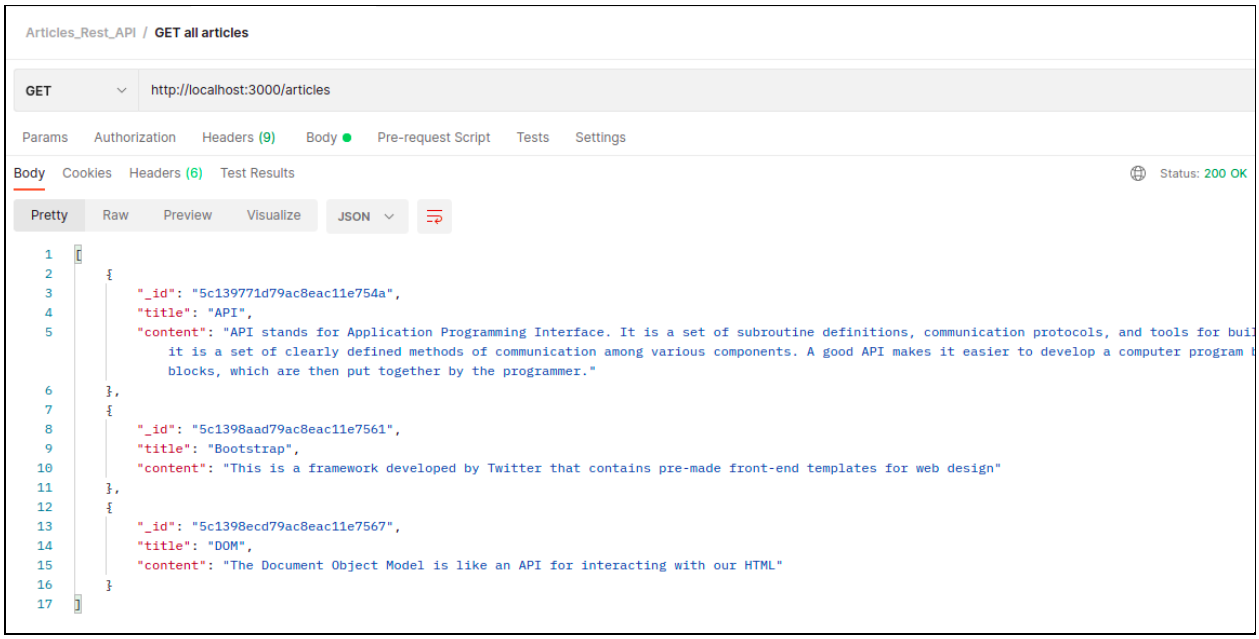
Database:

GET All articles:




```
app.get("/articles",function(request,response){
  Article.find(function(err,foundArticles){
    if(!err)
      response.send(foundArticles);
    else
      response.send(err);
  });
});
```

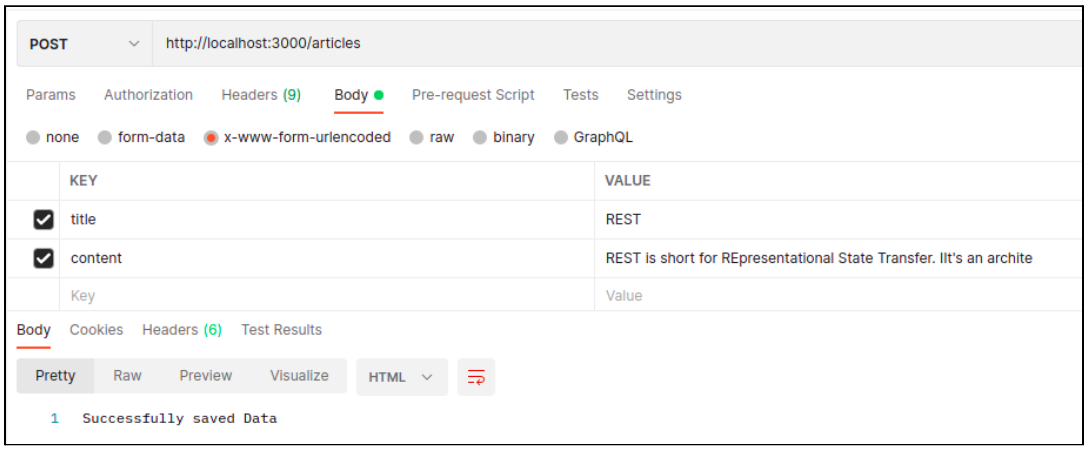
Output:

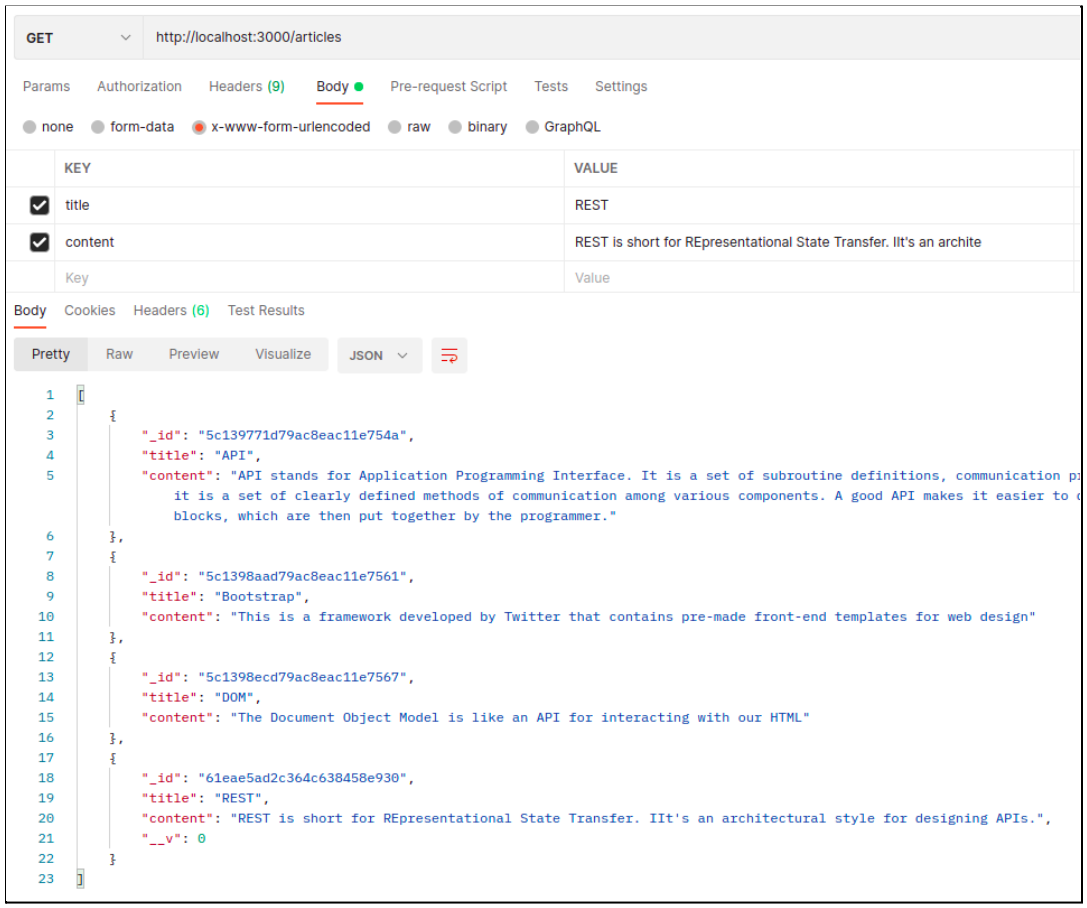


POST a new article:


```
app.post("/articles",function(request,response){
  const newArticle=new Article({
    title: request.body.title,
    content: request.body.content
  });
```

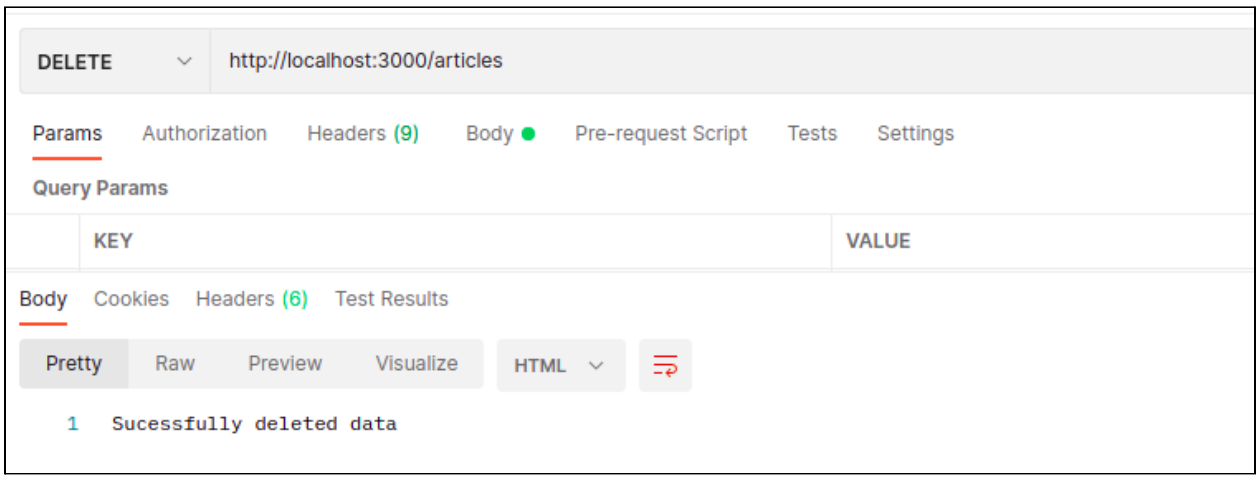
```
  newArticle.save(function(err){
    if(!err)
      response.send("Successfully saved Data");
    else
      response.send(err);
  });
});
```

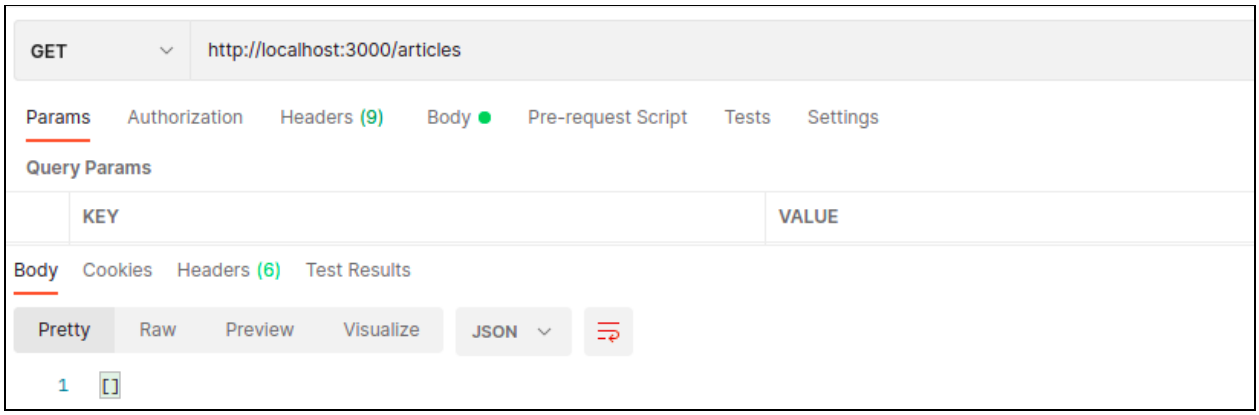




DELETE All articles:


```
app.delete("/articles",function(request,response){
  Article.deleteMany(function(err){
    if(!err)
      response.send("Successfully deleted data");
    else
      response.send(err);
  });
});
```

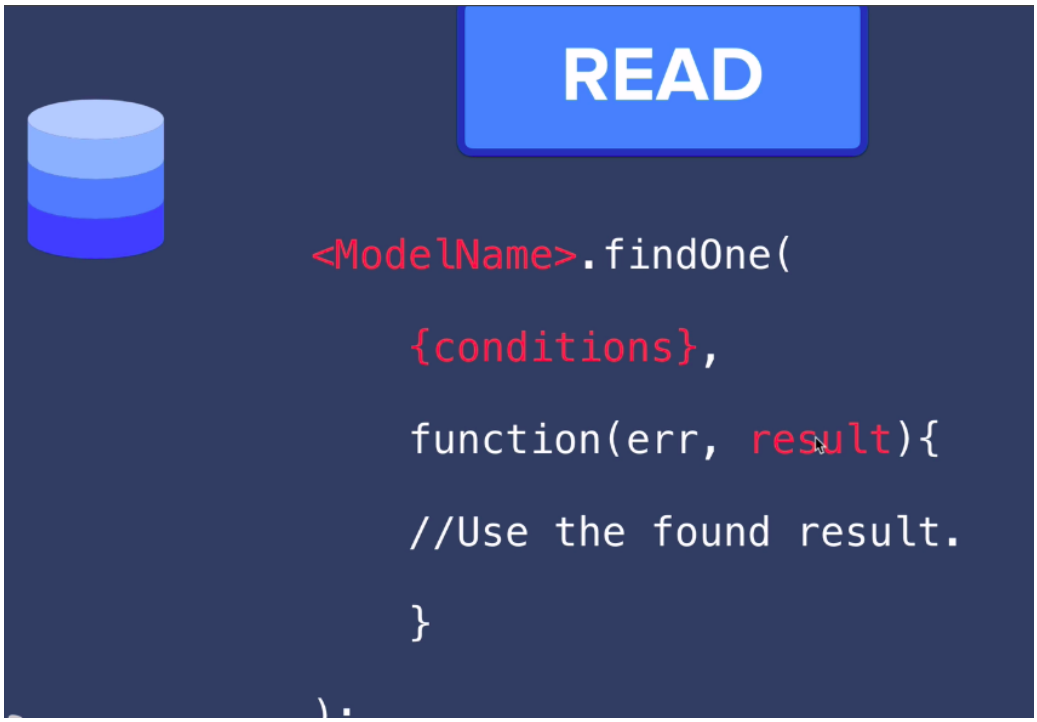





```
app.route("/articles")
.get(function(request,response){
  Article.find(function(err,foundArticles){
    if(!err)
      response.send(foundArticles);
    else
      response.send(err);
  });
})

.post(function(request,response){
  const newArticle=new Article({
```

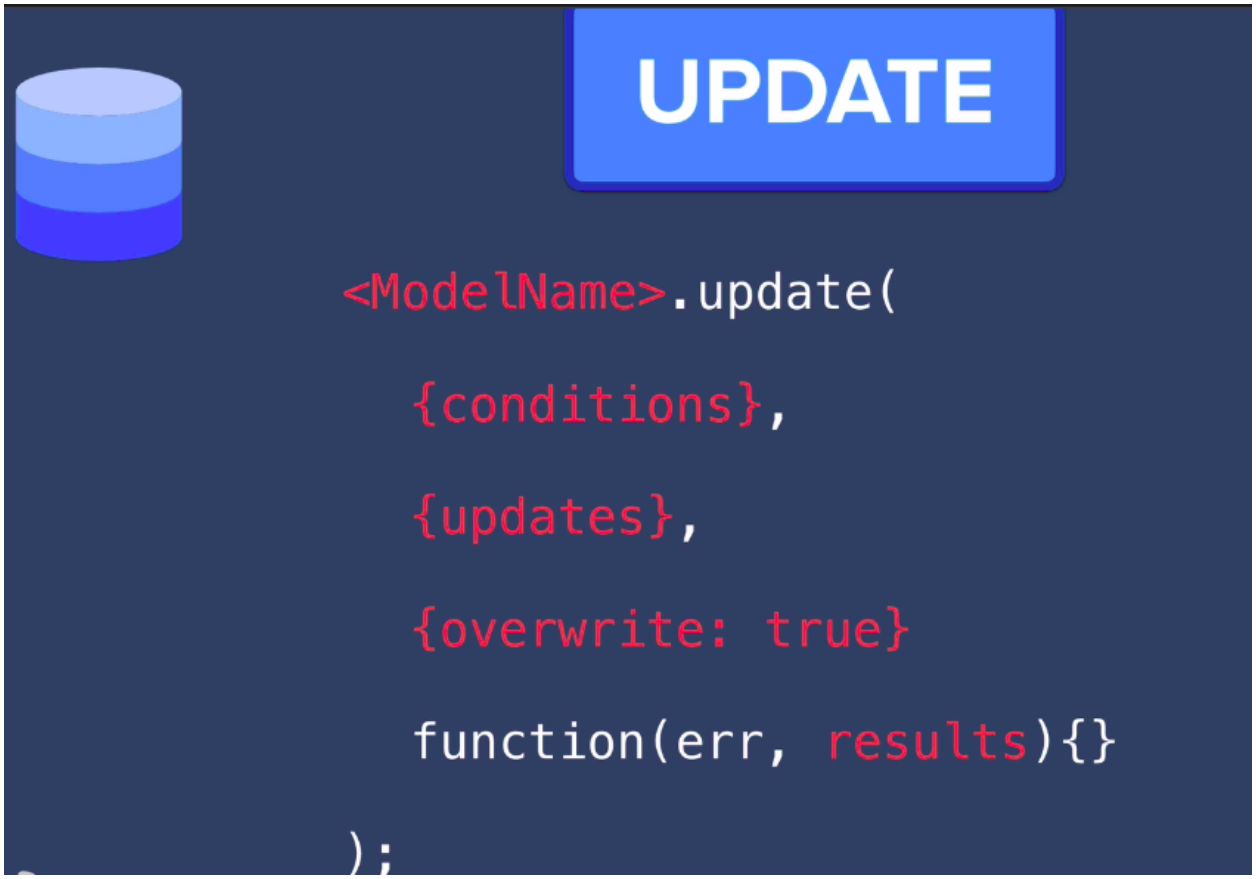
```
    title: request.body.title,
    content: request.body.content
  });
  newArticle.save(function(err){
    if(!err)
      response.send("Successfully saved Data");
    else
      response.send(err);
  });
})

.delete(function(request,response){
  Article.deleteMany(function(err){
    if(!err)
      response.send("Successfully deleted data");
    else
      response.send(err);
  });
});
```

GET a specific article:



```
app.route("/articles/:articleTitle")

.get(function(request,response){
  Article.findOne({title:
request.params.articleTitle},function(err,foundArticle){
    if(foundArticle)
      response.send(foundArticle);
    else
      response.send("No matching article found");
  });
})
```

PUT a specific article:




```
.put(function(request,response){
  Article.findOneAndUpdate(
    {title: request.params.articleTitle},
    {title: request.body.title,content: request.body.content},
    {overwrite:true},
    function(err){
    if(!err)
      response.send("Successfully updated data");
    else
      response.send(err);
  });
})
```

Articles_Rest_API / **PUT a specific article**

PUT http://localhost:3000/articles/DOM

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE
☑	title	DOM1
☑	content	Testing DOM
	Key	Value

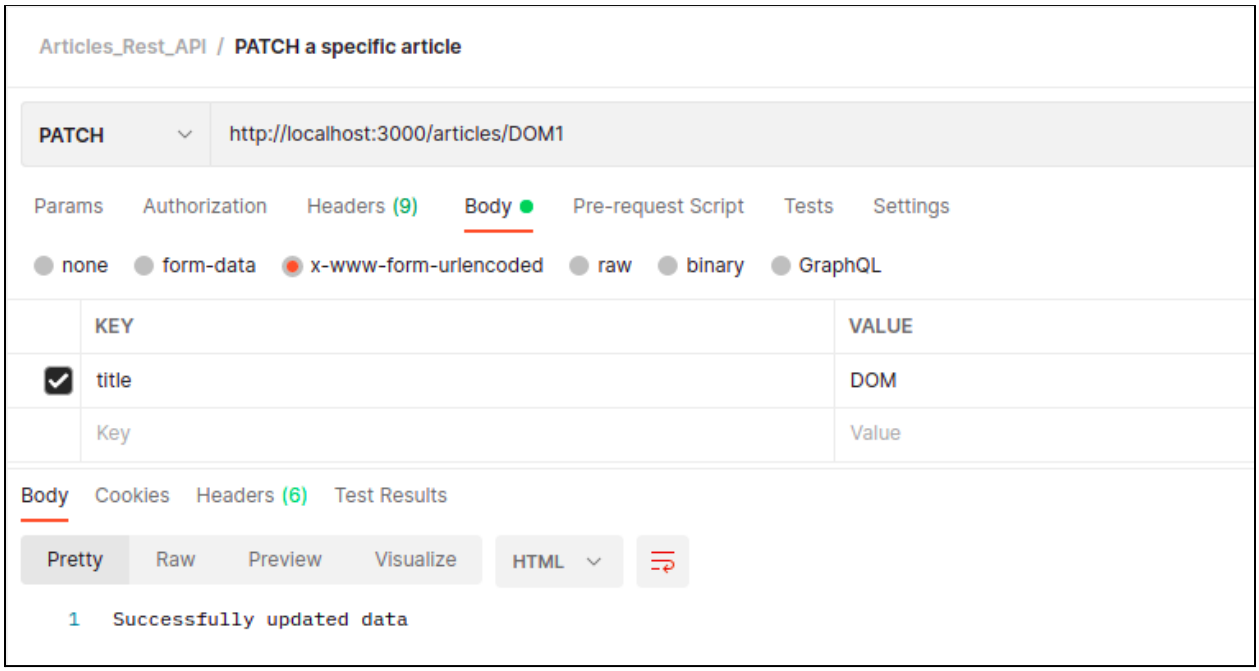
Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize HTML

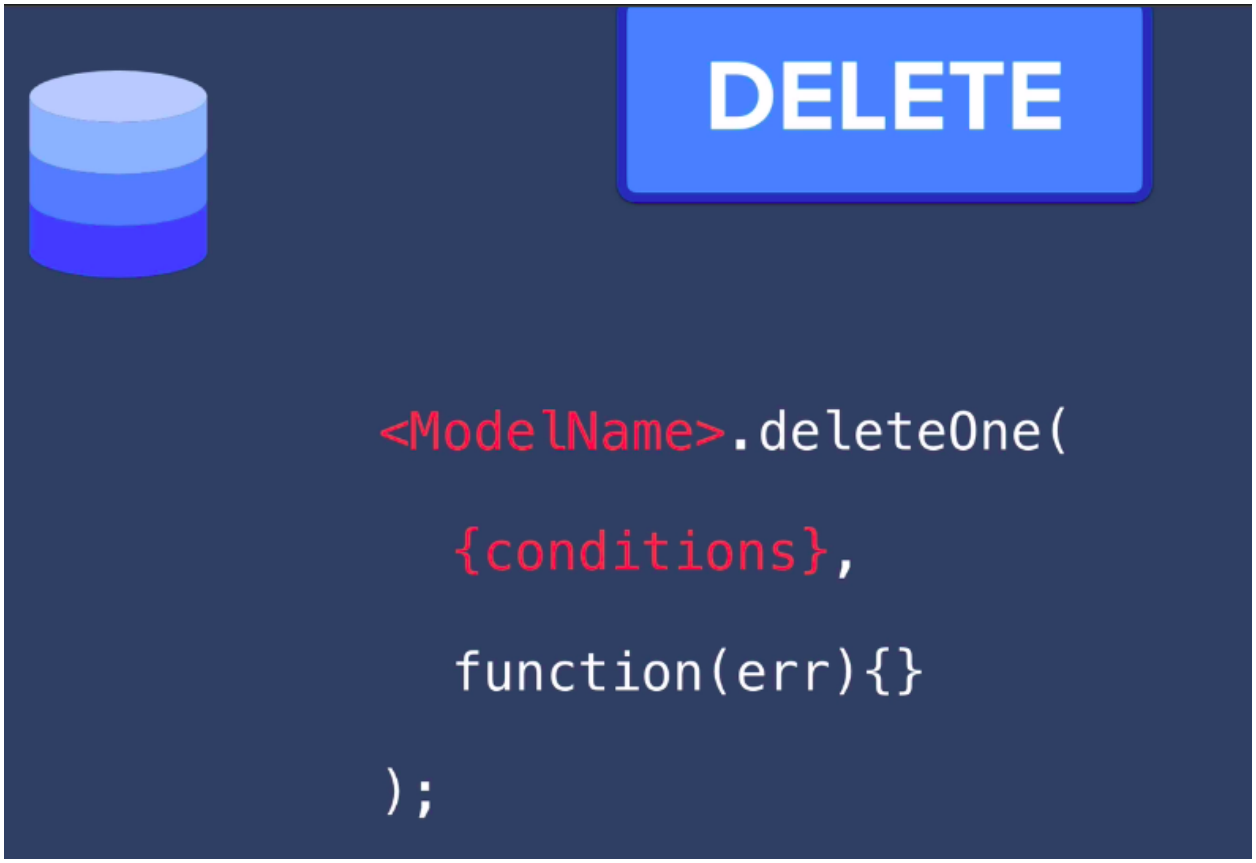
```
1  Sucessfully updated data
```

PATCH a specific article:

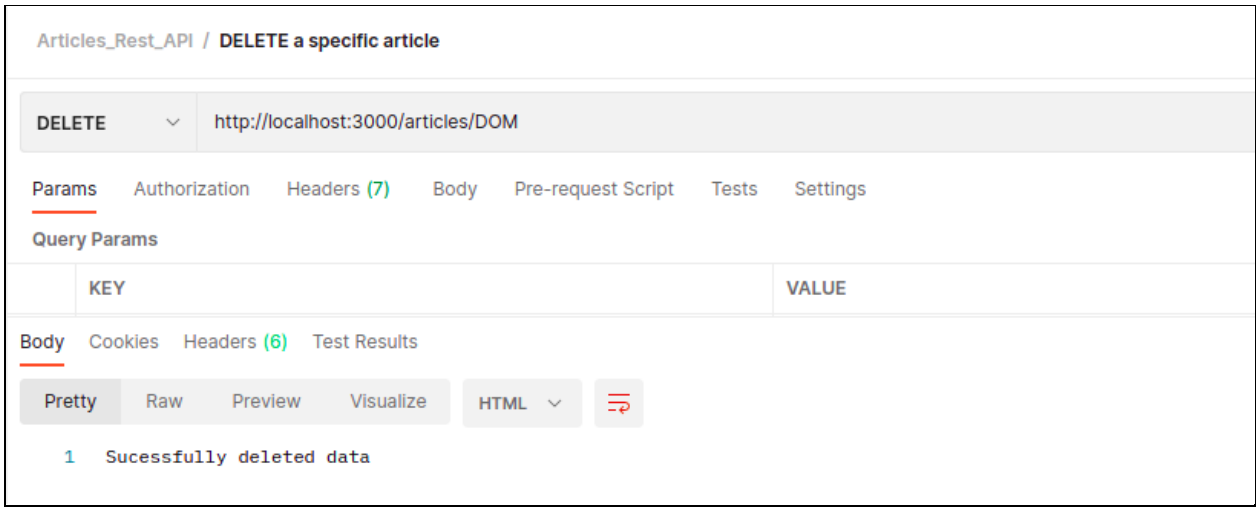
```
.patch(function(request,response){
  Article.findOneAndUpdate(
    {title: request.params.articleTitle},
    {$set: request.body},
    function(err){
    if(!err)
      response.send("Successfully updated data");
    else
      response.send(err);
  });
})
```



DELETE a specific article:




```
.delete(function(request,response){
  Article.deleteOne(
    {title: request.params.articleTitle},
    function(err,foundArticle){
    if(!err)
      response.send("Successfully deleted data");
    else
      response.send(err);
  });
});
```



References:

- What is an API URL Path? API URL Meaning [Explained] - Apipheny
- https://www.udemy.com/course/the-complete-web-development-bootcamp/learn/lecture/18125165#overview
- What Is REST API? | RESTful API Tutorial For Beginners | Edureka
- https://apipheny.io/what-are-api-parameters/
