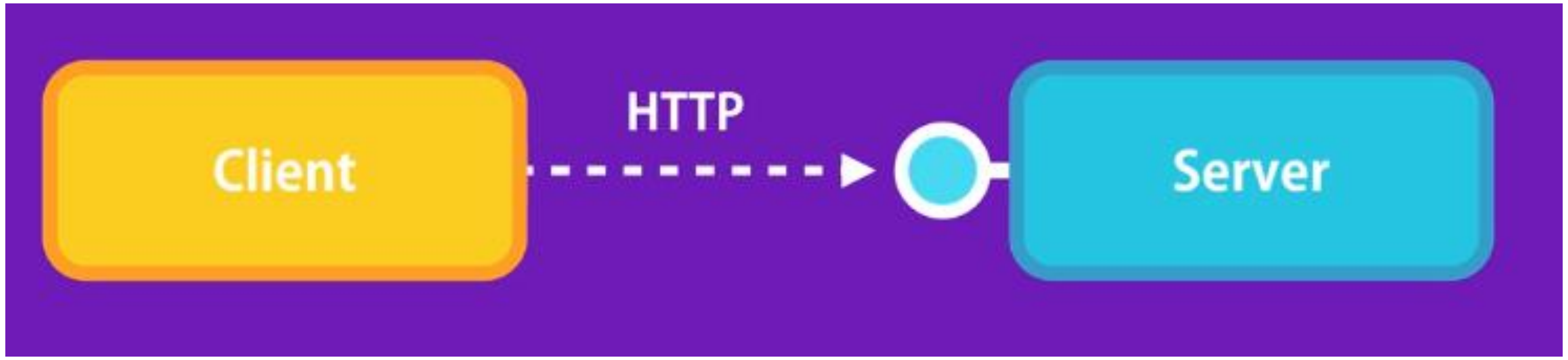
# NODE

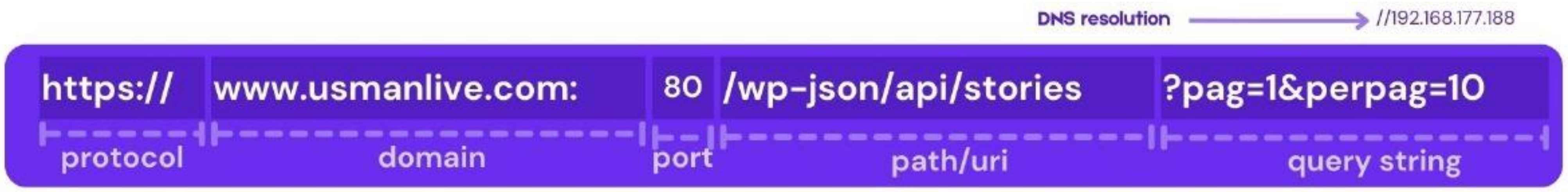
BUILDING RESTFUL API'S USING EXPRESS

## Client/Server Architecture

Client
Server

## Client/Server Architecture





## Client

### Web-Browser

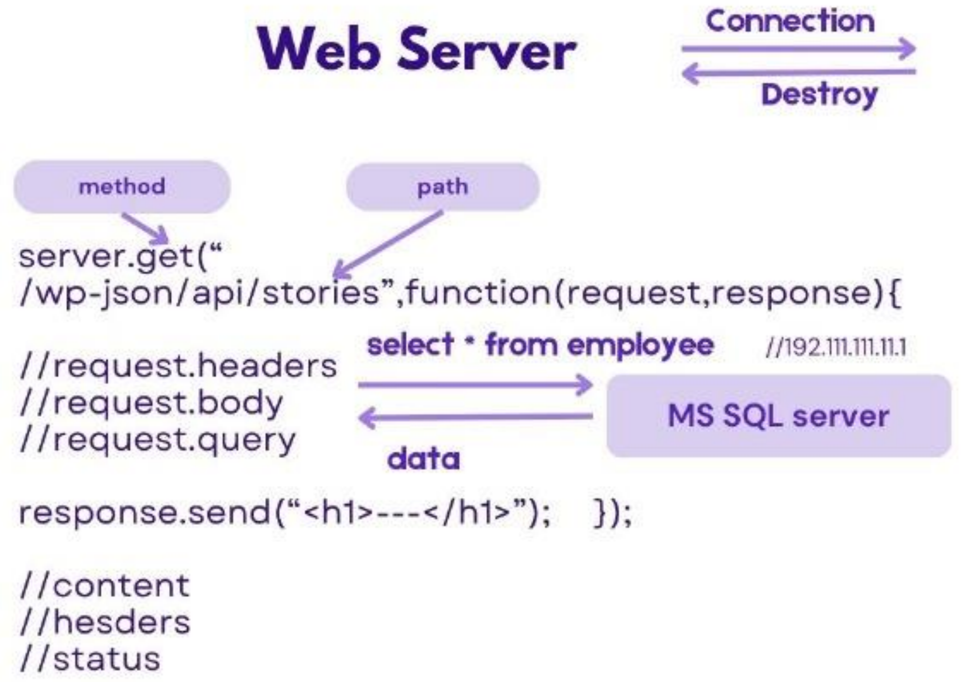
```
<form action="url">
<input name="email">
<input type="submit">
</form>
```

### Mobile Apps

```
-Android: 1) Kotlin
          2) Java
-ios     : 1) Swift
          2) objective C
-Both    : 1) React Native
          2) Fluttler
```

1. **url**
2. **method**: Get-default(post, put/patch, delete ....)
3. **header**
   Key1:value2
   Key2:value2
4. **body**

## Web Server



```
server.get("
/wp-json/api/stories",function(request,response){

//request.headers
//request.body
//request.query

response.send("<h1>---</h1>");    });

//content
//hesders
//status
```

## HTTP Request

### Request Method

- GET: Retrieve data from the server.
- POST: Submit data to be processed by the server.
- PUT: Update a resource on the server.
- DELETE: Remove a resource from the server.
- HEAD: Retrieve metadata about a resource, like its headers.
- OPTIONS: Inquire about the communication options available for a resource.
- PATCH: Apply partial modifications to a resource.

## HTTP Request

URI (Uniform Resource Identifier):

- This is a string that identifies the resource the client is requesting. It typically includes the web address (URL) and the path to the specific resource on the server.

## HTTP Request

### HTTP Version:

- Indicates the version of the HTTP protocol being used (e.g., HTTP/1.1).

### Headers:

- These are key-value pairs that provide additional information about the request or the client, such as the user agent (the type of browser or client making the request), the content type, and more.

### Body:

- In some types of requests, like POST or PUT, there may be data sent in the body of the request. For example, when submitting a form on a website, the form data is typically included in the request body.

## HTTP Request

Basic Authentication:
- Authorization: Basic base64encoded(username:password)

Bearer Token Authentication
- String sent in header

API Keys
- In headers or payload

Session Cookies:

Token-Based Authentication

JWT (JSON Web Token):

## Server Response

Status code

- (indicating the success or failure of the request),

headers,

response body (Optional)

- with data or content.

## Request Headers

**Host:** www.example.com

**User-Agent:**

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36

**Accept:** text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

## Response Headers

Content-Type: text/html; charset=utf-8

Content-Length: 12345

Server: Apache/2.4.51 (Unix)

## General Headers

Date: Tue, 25 Jan 2022 08:00:00 GMT

Connection: keep-alive

Cache-Control: max-age=3600

## Multipart Request

HTTP request that contains multiple parts

- each with its own set of headers and body

commonly used when submitting forms that include files

# GET Vs POST

| GET | POST |
|---|---|
| Request data | Submit data |
| Data sent to server in url | Data sent to server in body |
| Used for operations where data modification isn’t required | Used for data modifications operations |
| cached by browsers and servers | not cached by browsers or servers |
| Are generally book marked | Not Bookmarakable |

## Pain to code like this

```
 const server =
http.createServer((req,res)=>{
  if(req.url==='/'){
  res.write(JSON.stringify([1,4,5,6]));
  res.end();
  }
 });
```

## Express https://www.npmjs.com/package/express

15 Million Downloads a Month

```
npm i express
```

## Express

```
const express = require('express');
const app = express();
app.use(express.json()); //middleware
//handle api calls here
```

## Express

```
const port = 3000;
app.listen(port,function(){
console.log(`Listening on Port 3000....`);
})
```

## localhost:3000/api/greet

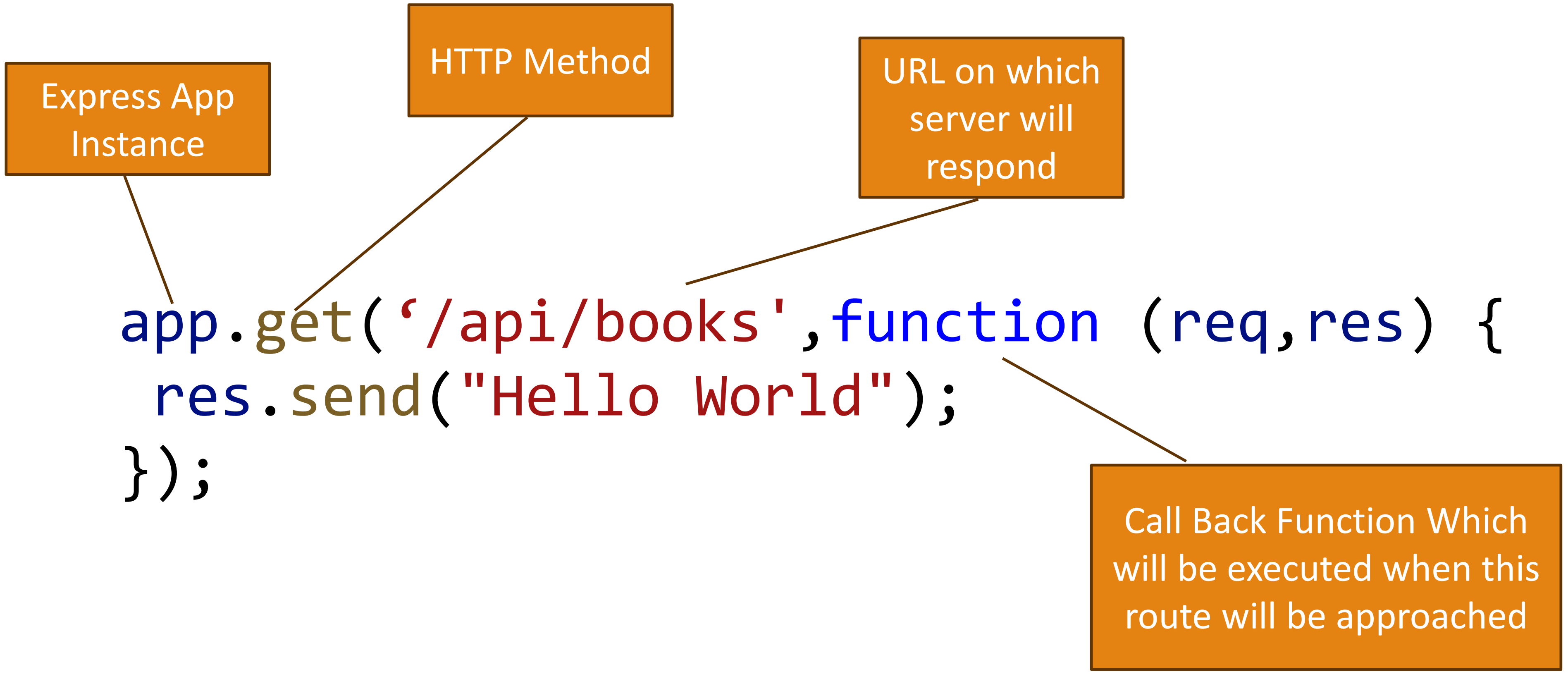
```
const express = require('express');
const app = express();
// Sample route to handle GET requests
app.get('/api/greet', (req, res) => {
  res.json({ message: 'Hello from the Express API!' });
});
// Start the server
app.listen(3000, () => {
  console.log(`Express server listening on port ${port}`);
});
```

A Simple Server to listen on port 3000 and respond to a one single call.
Note: it does not send back the html rather a JSON Object

## Get call

```
http method: get, post, put,delete
handler: a function with two inputs Request and  Response

url: /api/books
```

```
app.get(‘/api/books',function (req,res) {
 res.send("Hello World");
});
```


Express App Instance
HTTP Method
URL on which server will respond
Call Back Function Which will be executed when this route will be approached

```
app.get(‘/api/books',function (req,res) {
 res.send("Hello World");
});
```

## Route Handler app.METHOD(PATH, HANDLER);

```
// Define a route with a callback function
app.get('/hello', (req, res) => {
  res.send('Hello, World!');
});
```

The get method is used to define a route for handling HTTP GET requests.

The route path is /hello.

The callback function **(req, res) => { res.send('Hello, World!'); }** is the handler that will be executed when a GET request is made to /hello. The req parameter represents the request object, and res represents the response object.

## Call Back Function Inputs

- **req (request):**
  - An object representing the incoming HTTP request, containing information about the client's request.
- **res (response):**
  - An object representing the server's response, allowing you to send data back to the client.
- **Next (Optional Function):**
  - The next function, when called, passes control to the next middleware function in the stack.

## Send back html

```
app.get('/sample-html', (req, res) => {
  const htmlString = `
     <html lang="en">
      <body>
        <h1>Hello, this is a sample HTML page!</h1>
        </body>
    </html>
  `;
// Send the HTML string as the response
  res.send(htmlString);
});
```

We can send back html as response
Sound like a proper web server which send back html

## REpresentational State Transfer (REST)

- Architectural style for designing networked applications
  - a set of principles
- Statelessness
- Client-Server Architecture
- Uniform Interface:
  - **Resource Identification:** Resources (entities or services) are uniquely identified by URIs (Uniform Resource Identifiers).
- used in web development for building scalable and interoperable web services

## RESTFUL API Design Requirements

Client Server Architecture

Statelessness

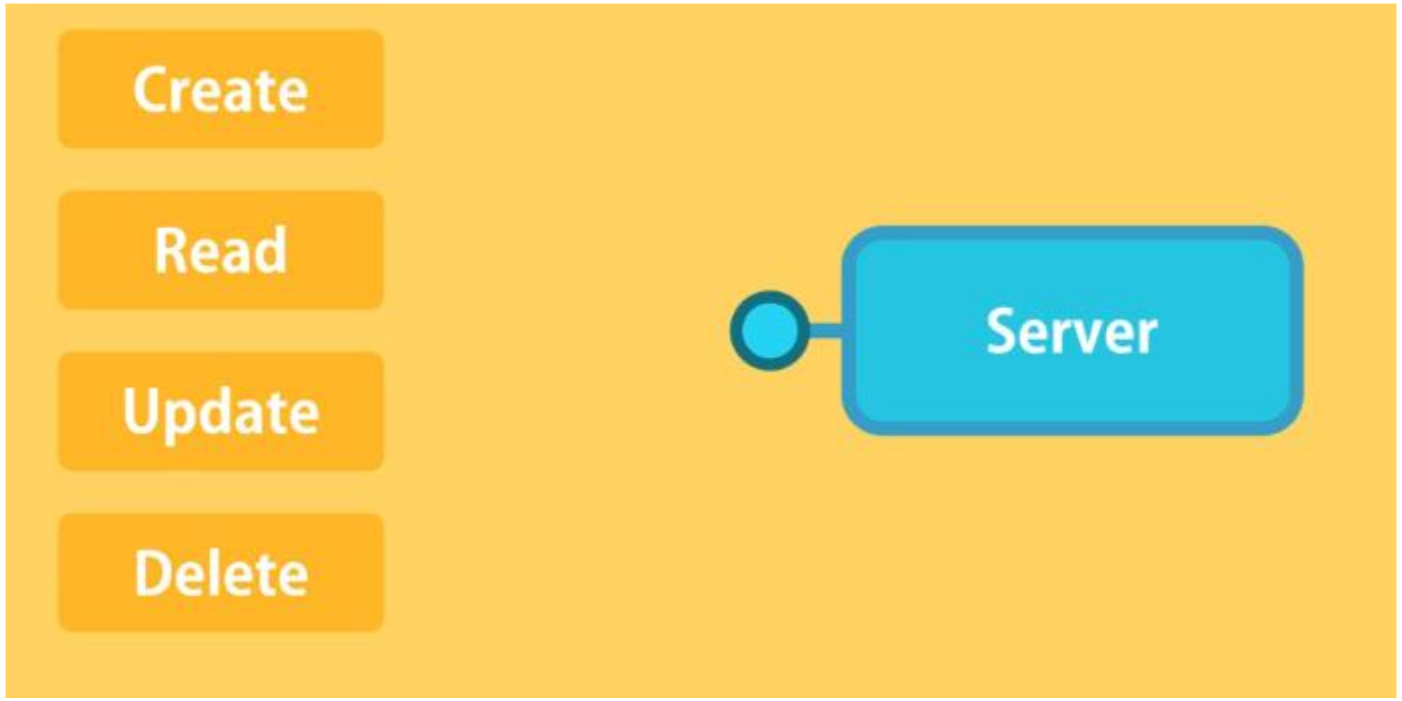
Uniform Interface

- Resource Based
- Representation

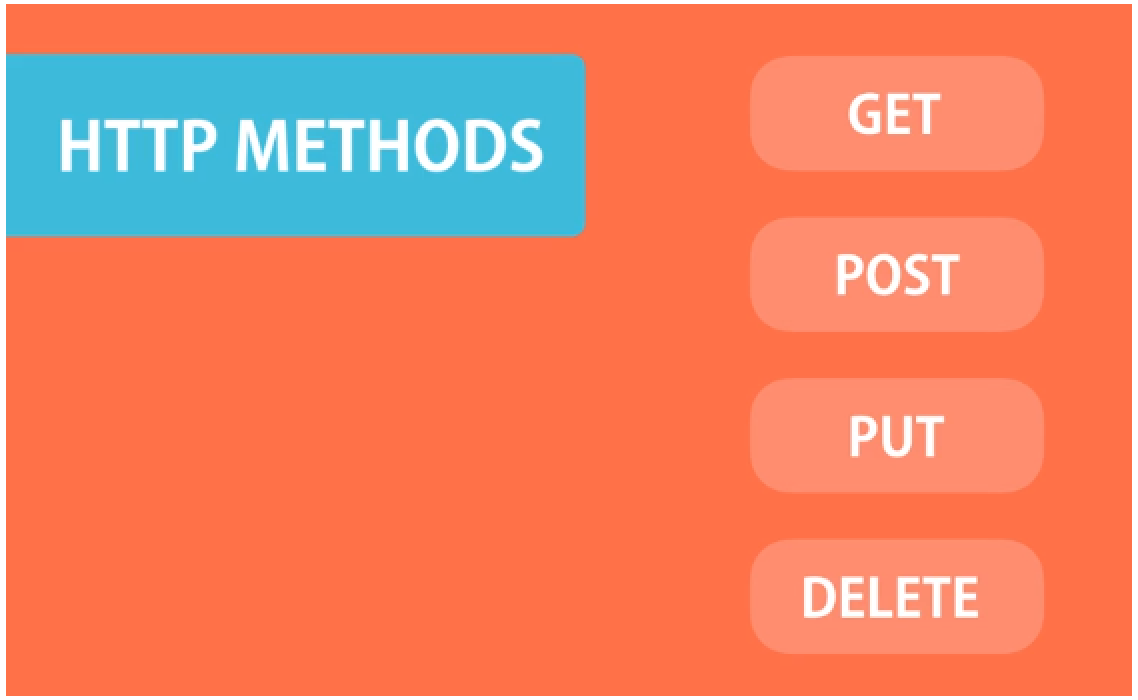
Cacheability

Layered System

# <u>RE</u>presentational <u>S</u>tate <u>T</u>ransfer (REST)



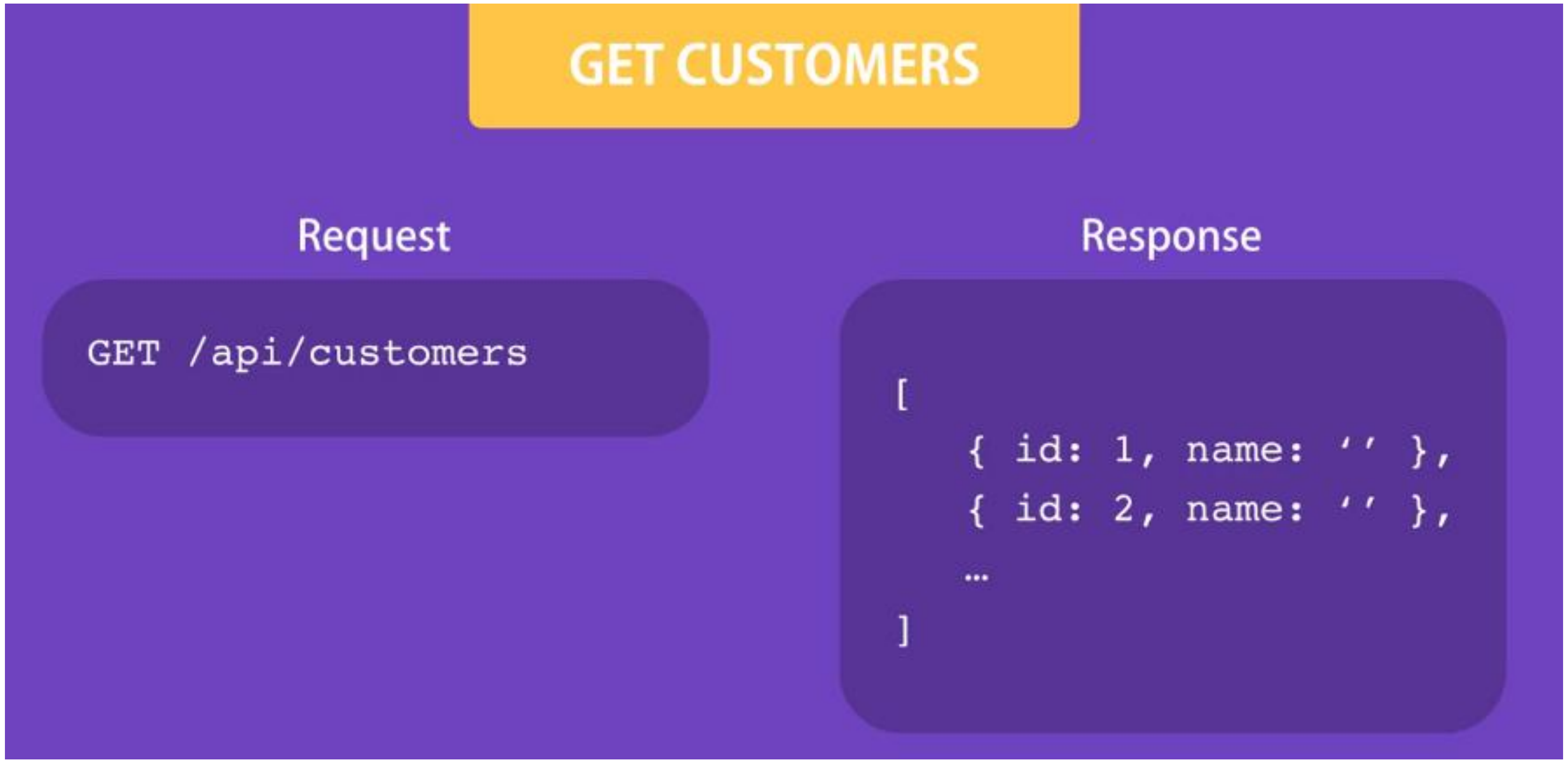
# http://usman.com/api/customers



| HTTP Method | Db Query |
|---|---|
| GET | Select |
| POST | Insert |
| Put/Patch | Update |
| Delete | Delete |

You can only use GET and POST with browser

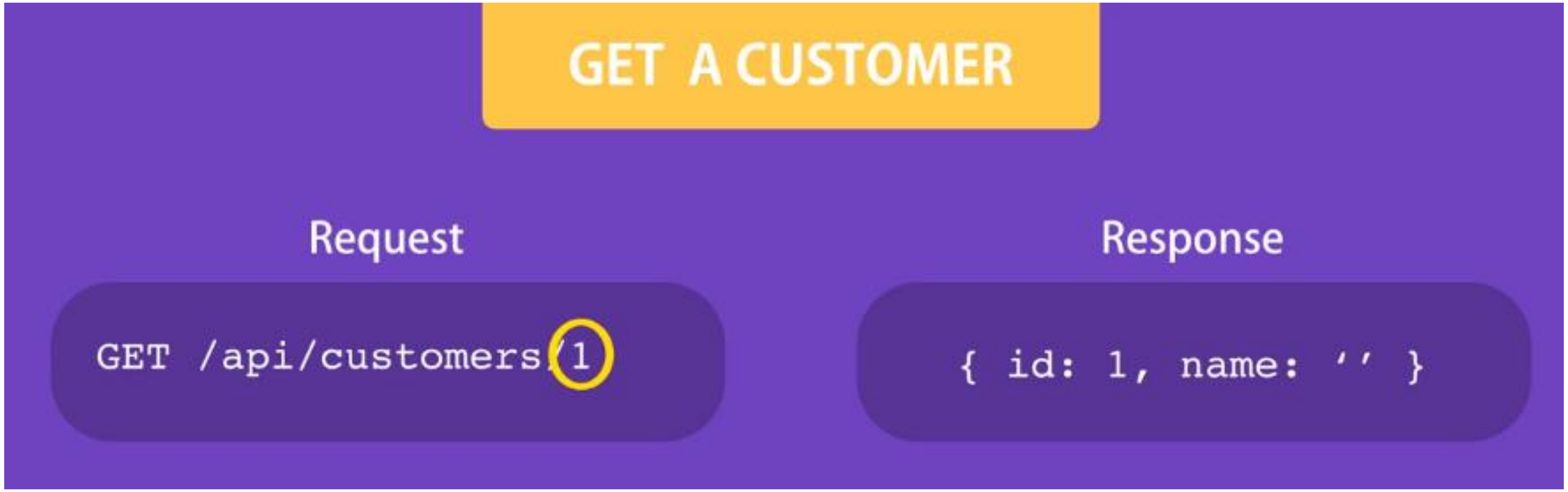
## http://usman.com/api/customers



```
GET /api/customers
```

```
[
    { id: 1, name: '' },
    { id: 2, name: '' },
    ...
]
```

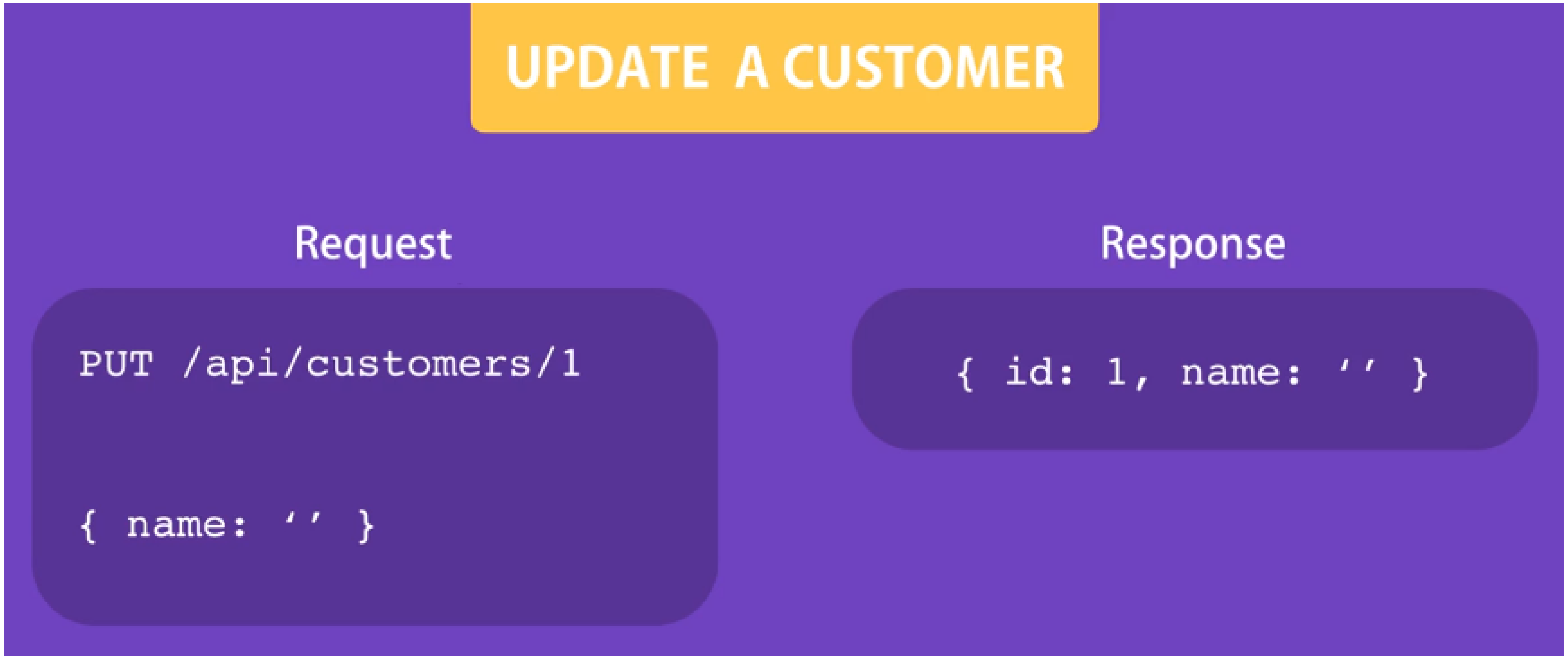
## http://usman.com/api/customers/1



```
GET /api/customers/1
```

```
{ id: 1, name: '' }
```

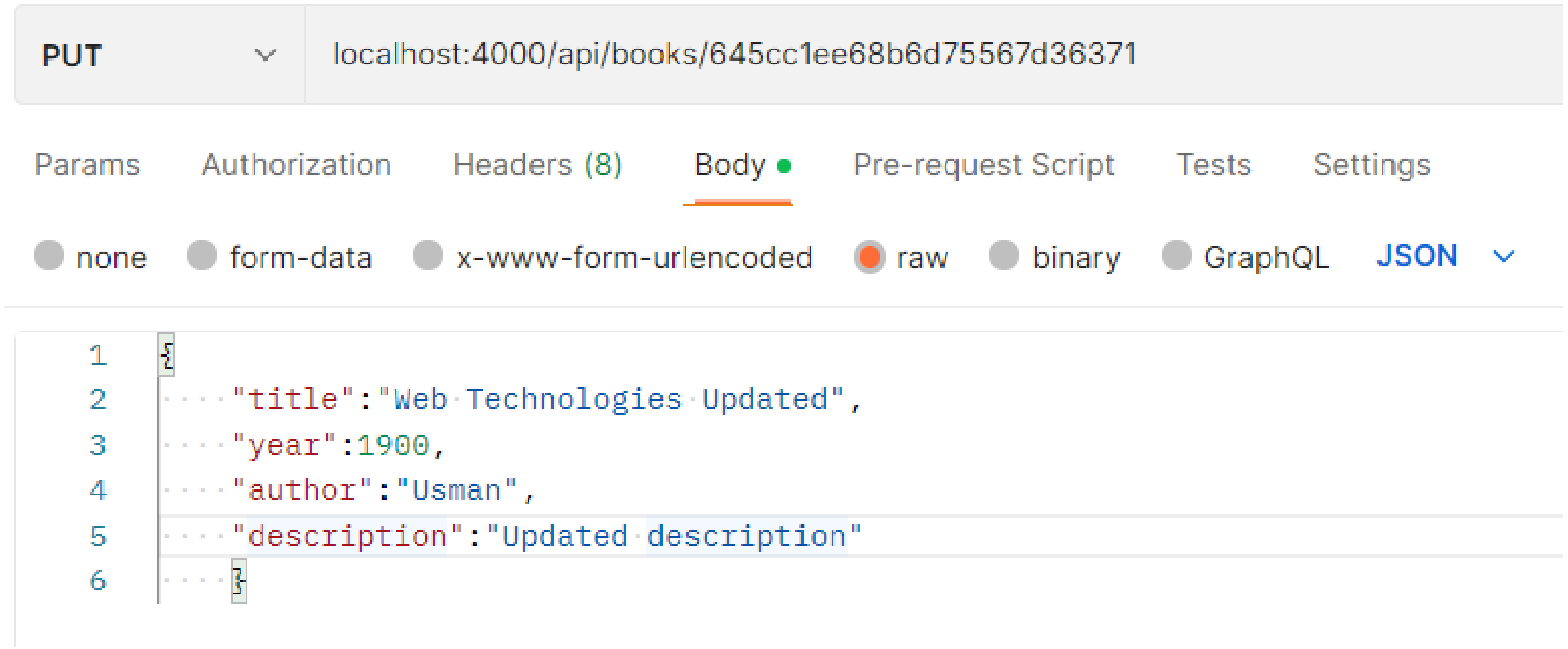
# http://usman.com/api/customers/1



```
PUT /api/customers/1

{ name: '' }
```

```
{ id: 1, name: '' }
```

## Use POSTMan to send put request

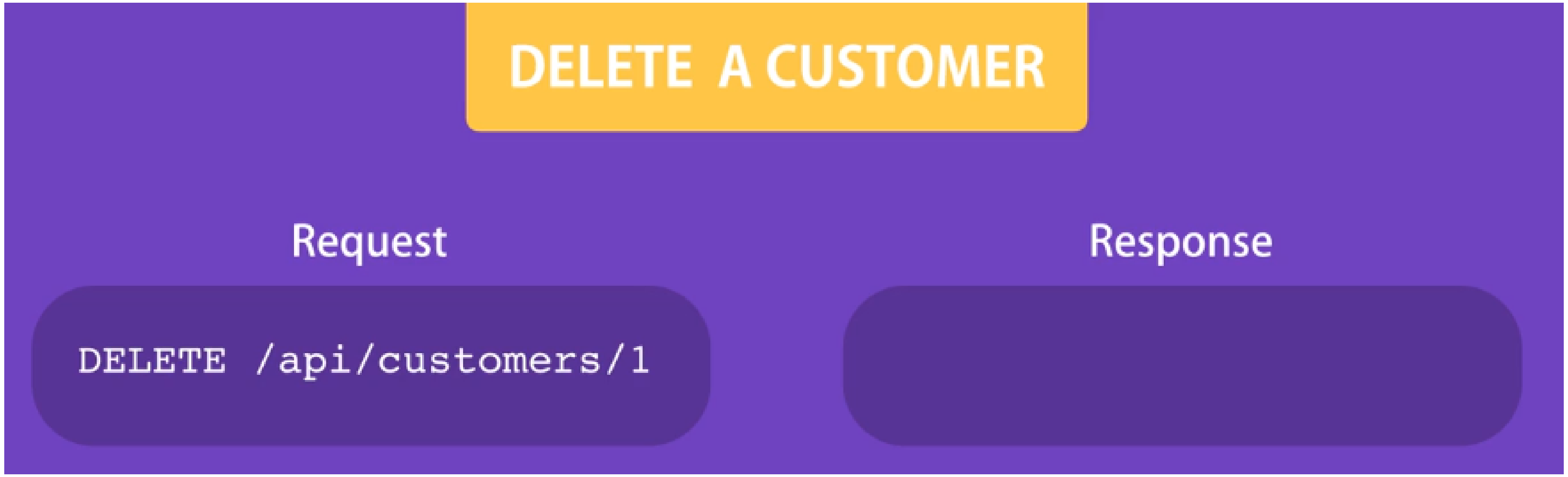

PUT localhost:4000/api/books/645cc1ee68b6d75567d36371

Params | Authorization | Headers (8) | Body ● | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

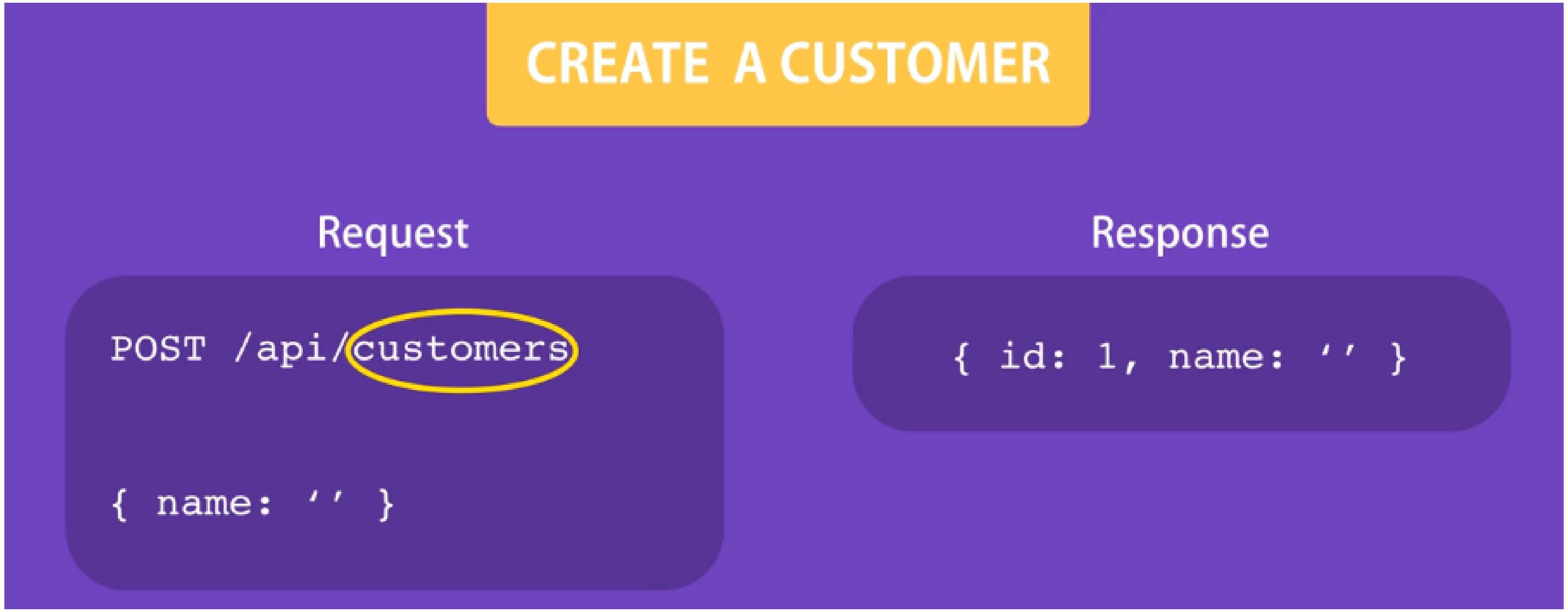
```
{
    "title":"Web Technologies Updated",
    "year":1900,
    "author":"Usman",
    "description":"Updated description"
    }
```

## http://usman.com/api/customers/1



```
DELETE /api/customers/1
```

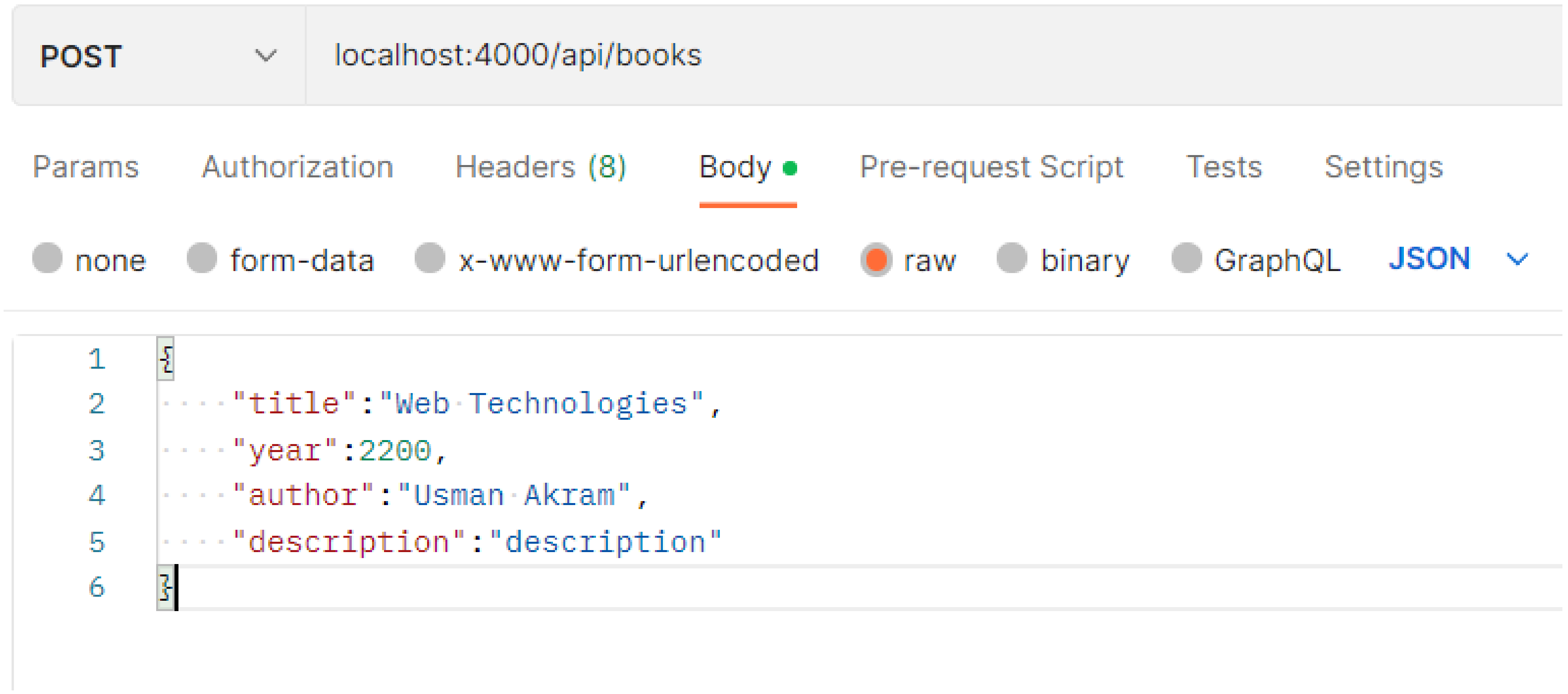
# http://usman.com/api/customers



```
POST /api/customers

{ name: '' }
```

```
{ id: 1, name: '' }
```

# Post Request from postman

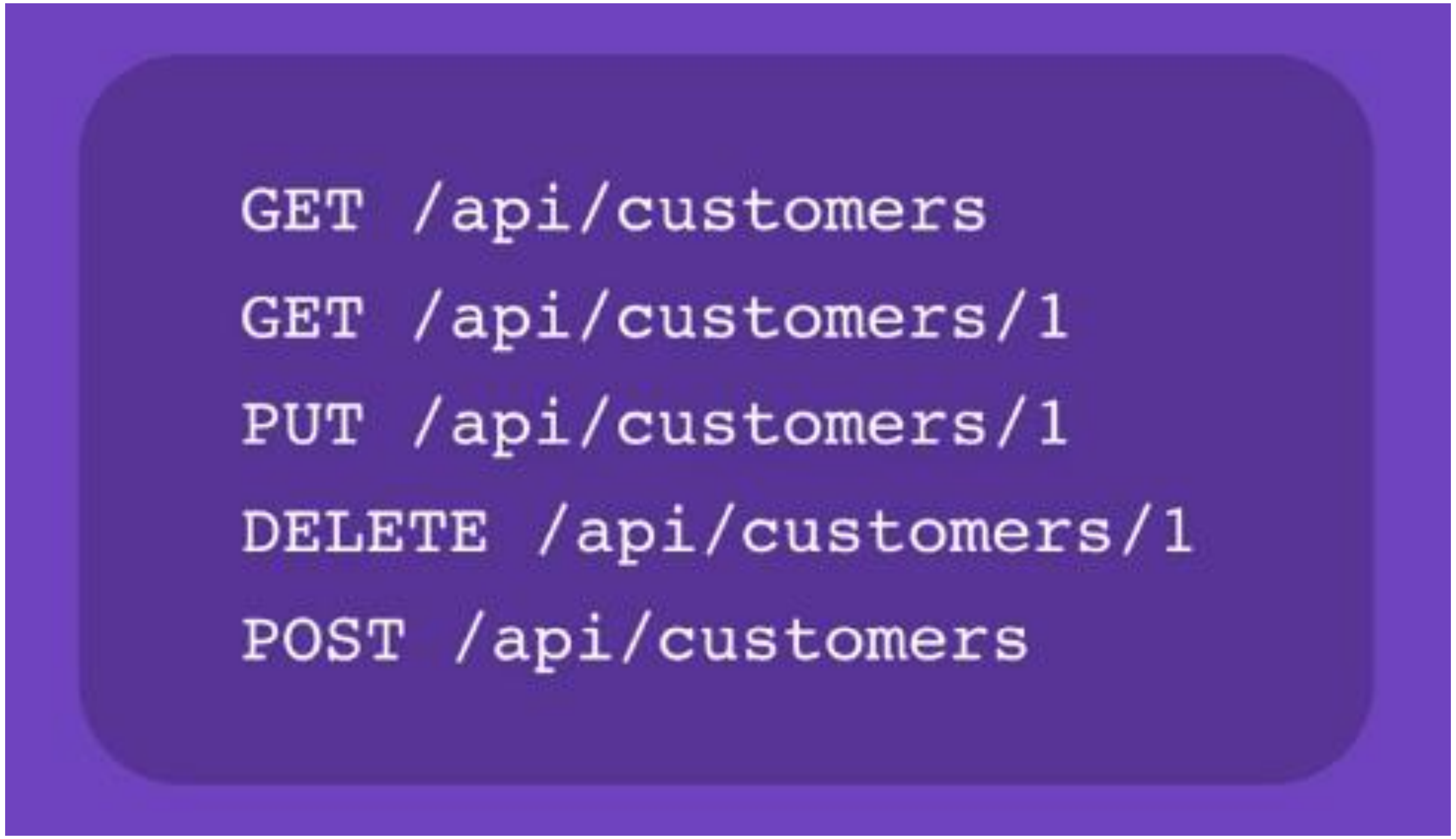


localhost:4000/api/books

Body: [✓ raw] JSON

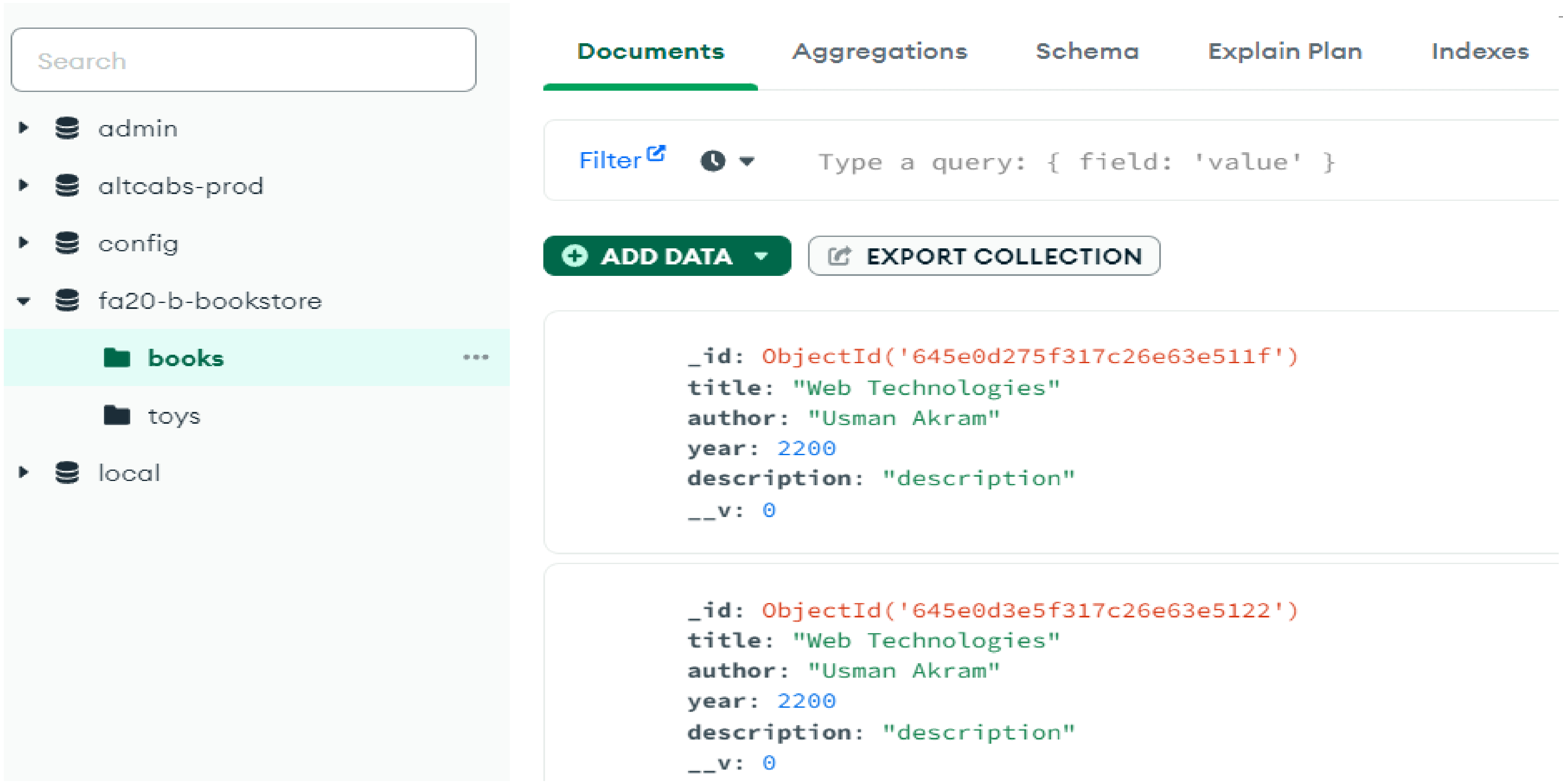
```json
{
    "title":"Web Technologies",
    "year":2200,
    "author":"Usman Akram",
    "description":"description"
}
```

## All RESTFUL calls



```
GET /api/customers
GET /api/customers/1
PUT /api/customers/1
DELETE /api/customers/1
POST /api/customers
```

# Mongo DB



```
_id: ObjectId('645e0d275f317c26e63e511f')
title: "Web Technologies"
author: "Usman Akram"
year: 2200
description: "description"
__v: 0
```

```
_id: ObjectId('645e0d3e5f317c26e63e5122')
title: "Web Technologies"
author: "Usman Akram"
year: 2200
description: "description"
__v: 0
```

## Connecting to Mongo use mongoose

```
const mongoose = require('mongoose');
mongoose.connect("mongodb://localhost/diabudy",
{ useNewUrlParser: true })
.then(() => console.log("Connected to Mongo
...."))
.catch((error) => console.log(error.message));
```

## Create a Schema

```
let bookSchema = mongoose.Schema({
  title: String,
  author: String,
  year: Number,
  description: String,
});
```

## Define Model

```
const Book = mongoose.model("Book", bookSchema);
```

## /models/category.js

```
let mongoose = require("mongoose");

let schema = new mongoose.Schema({
  title: String,
});

let Model = mongoose.model("Category", schema);
 module.exports = Model;
```

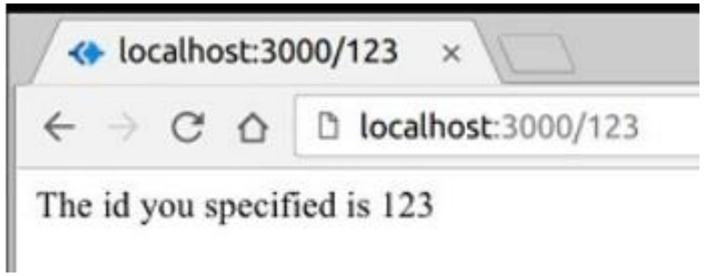
## Send array back to client

```
app.get("/api/books", async (req, res) => {
  let books = await Book.find();
  res.send(books);
});
```

Book.find() is actually an I/O call to database server. So we stop code execution by using await until query is executed at db and returned. We will study async/await later on

## Route Parameter

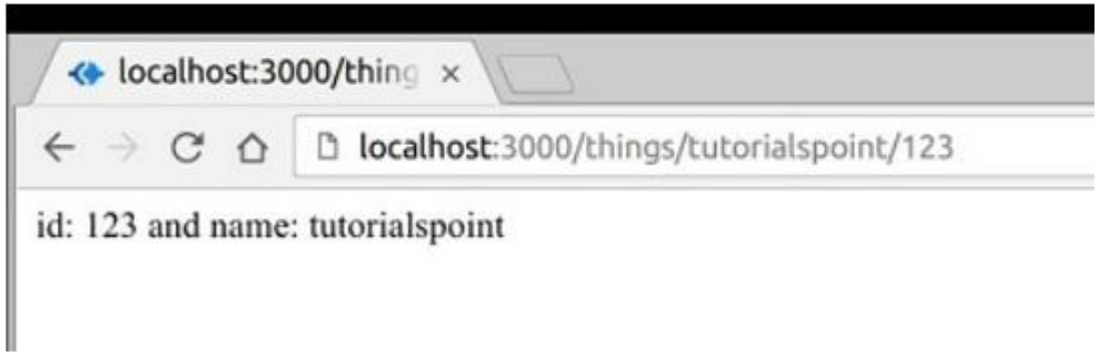
```
var express = require('express');
var app = express();

app.get('/:id', function(req, res){
   res.send('The id you specified is ' + req.params.id);
});
app.listen(3000);
```



The id you specified is 123

## Multiple Parameters

```
app.get('/things/:name/:id', function(req, res) {
   res.send('id: ' + req.params.id + ' and name: ' + req.params.name);
});
```



```
id: 123 and name: tutorialspoint
```

## Get single record (Receive id in route parameters)

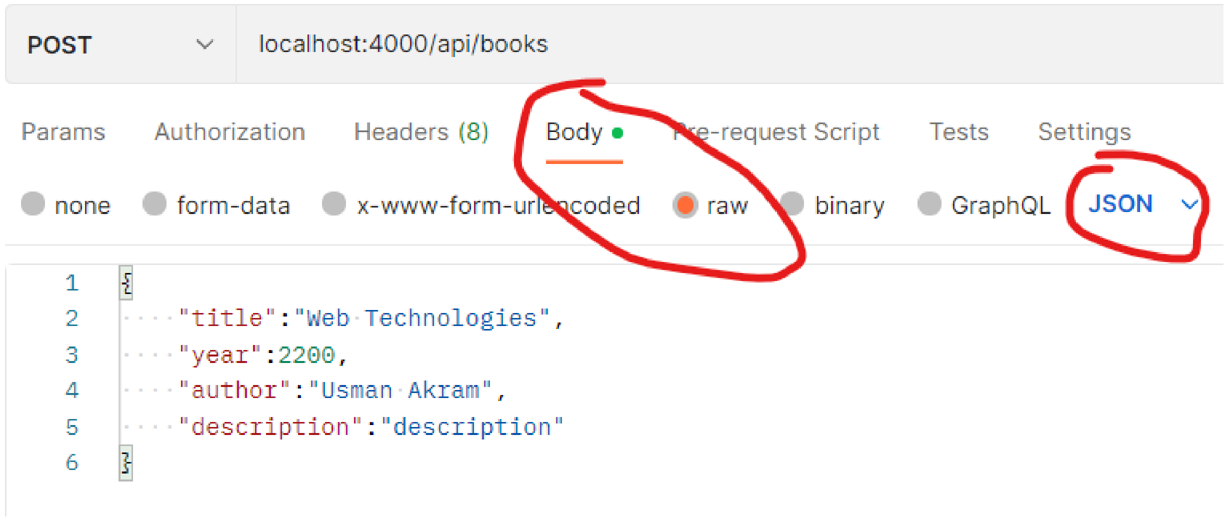
```
 router.get(“/api/books/:id", async (req, res) =>
{
   let book = await Book.findById(req.params.id);
   res.send(book);
 });
```

## Delete a Record

```
router.delete("/api/books/:id", async (req, res) => {
   let book = await Book.findByIdAndDelete(req.params.id);

  res.send(book);
});
```

## Create a Record. Data is sent via json in request body

```
 router.post(“/api/books", async (req, res)
=> {
   let newBook = new Book(req.body);
   await newBook.save()
   res.send(newBook);
 });
```

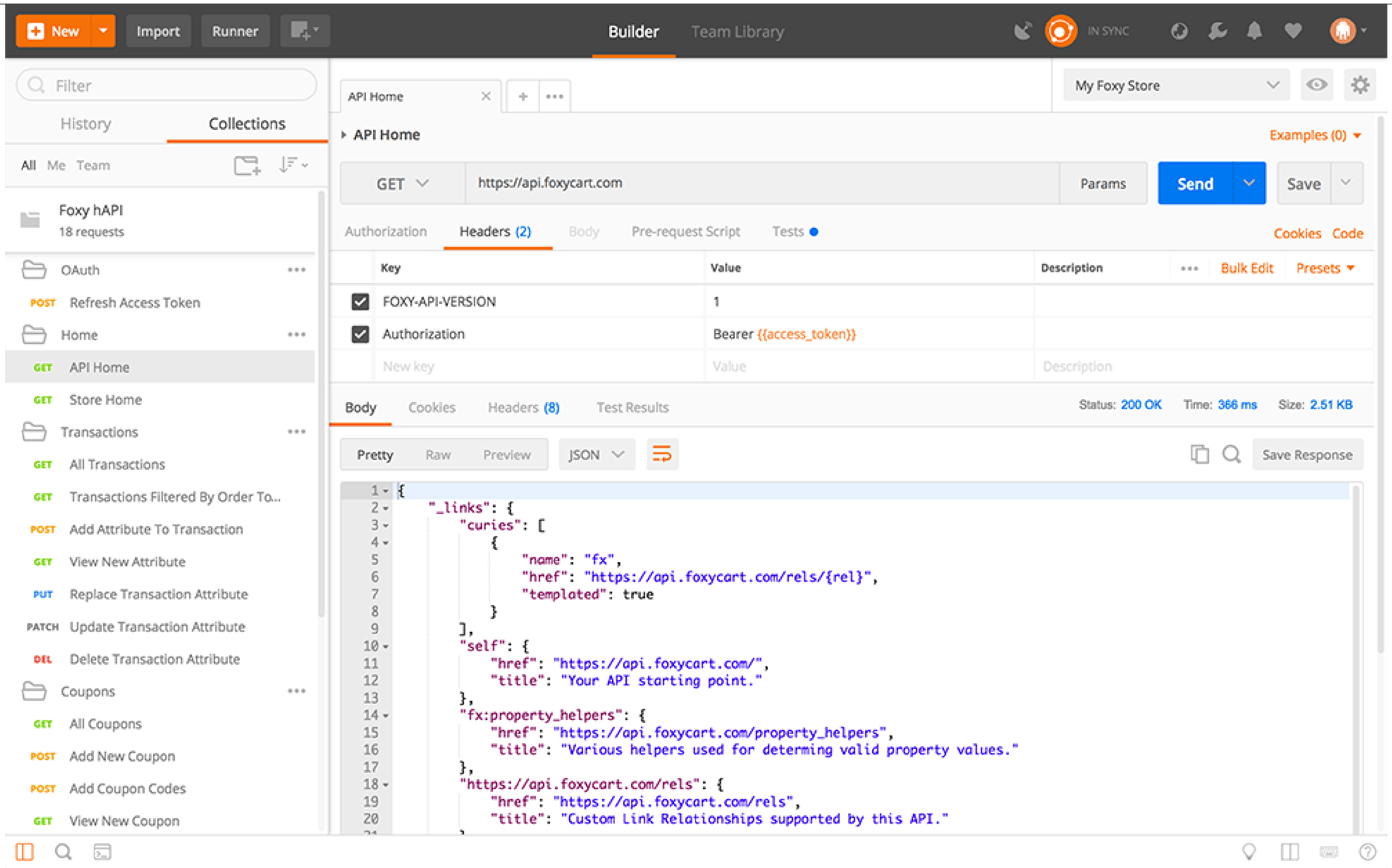


localhost:4000/api/books

[✓ raw] [✓ JSON]

```
{
    "title":"Web Technologies",
    "year":2200,
    "author":"Usman Akram",
    "description":"description"
}
```

# Postman



Foxy hAPI
18 requests

- OAuth
  - POST Refresh Access Token
- Home
  - GET API Home
  - GET Store Home
- Transactions
  - GET All Transactions
  - GET Transactions Filtered By Order To...
  - POST Add Attribute To Transaction
  - GET View New Attribute
  - PUT Replace Transaction Attribute
  - PATCH Update Transaction Attribute
  - DEL Delete Transaction Attribute
- Coupons
  - GET All Coupons
  - POST Add New Coupon
  - POST Add Coupon Codes
  - GET View New Coupon

GET https://api.foxycart.com

| | Key | Value | Description |
|---|---|---|---|
| ✓ | FOXY-API-VERSION | 1 | |
| ✓ | Authorization | Bearer {{access_token}} | |
| | New key | Value | Description |

Status: 200 OK Time: 366 ms Size: 2.51 KB

```
{
    "_links": {
        "curies": [
            {
                "name": "fx",
                "href": "https://api.foxycart.com/rels/{rel}",
                "templated": true
            }
        ],
        "self": {
            "href": "https://api.foxycart.com/",
            "title": "Your API starting point."
        },
        "fx:property_helpers": {
            "href": "https://api.foxycart.com/property_helpers",
            "title": "Various helpers used for determing valid property values."
        },
        "https://api.foxycart.com/rels": {
            "href": "https://api.foxycart.com/rels",
            "title": "Custom Link Relationships supported by this API."
```

## express.json()
## Handles JSON Data

Built-in middleware function in Express.

It parses incoming requests with JSON payloads

Based on body-parser.

Add below line to include it.

```
app.use(express.json());
```

## express.urlencoded()
## Handles Form Data

Built-in middleware function in Express.

It parses incoming requests with URL-encoded payloads

Based on body-parser.

Add below line to include it.

The "extended" syntax allows for rich objects and arrays to be encoded into the URL-encoded format,

```
app.use(express.urlencoded({ extended: false }));
```

## Route parameters

```
app.put(“/api/books/:id", async (req, res) => {
  let book = await Book.findById(req.params.id);
  book.title = req.body.title;
  book.year = req.body.year;
  book.author = req.body.author;
  book.description = req.body.description;
  await book.save();
  res.send(book);
});
```

## Validation with joi

npm I joi

```
const Joi = require('joi');
const schema = {
name:Joi.string().min(3).required()
}
```

## Validation with joi

```
const result =
Joi.validate(request.body,schema);

//console.log(result);

if(result.error)
response.status(400).send(result.error.details[0].message);
```

## Clean joi

```
function validateCourse(course) {
const schema = {
name: Joi.string().min(3).required()
};
const result = Joi.validate(course, schema);
return result;
} //require joi at top preferably extract a
//module
```

# Express- Advanced Topics

GO PRO

## Middleware

```
app.get('/',function (request,response) {
response.send("Hello World");
});
```

Is technically a middleware. It breaks the request response cycle

What if we do something here and then don’t send response.

## Adding a middleware

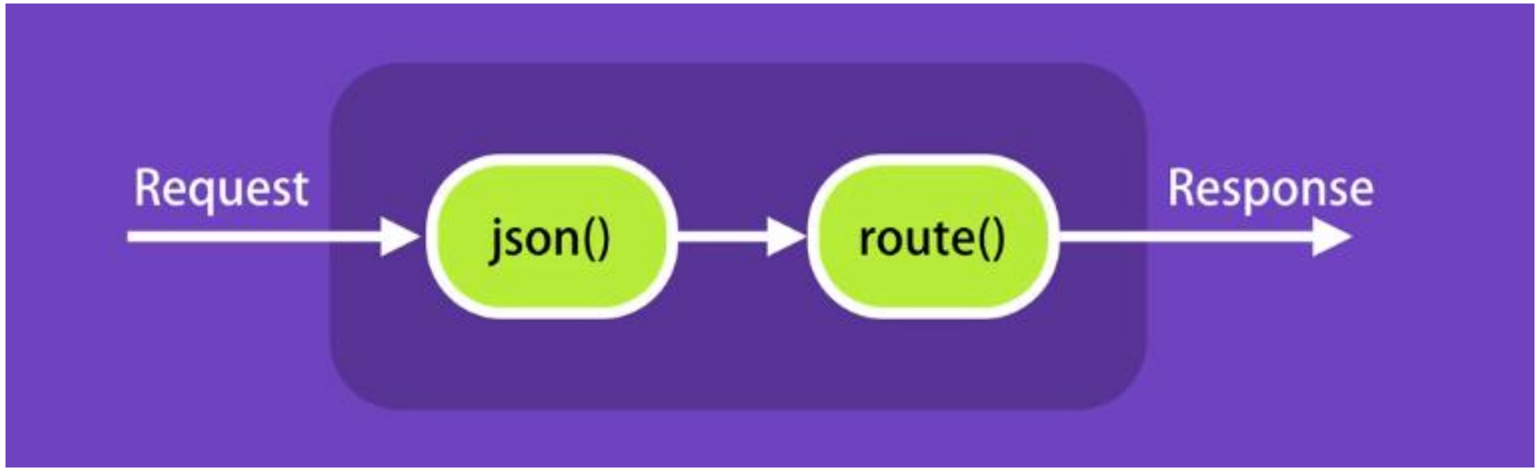
```
const express = require('express');
const app = express();
app.use(express.json());
```

Get request

If body contain json then parse it

Forward it to next middleware.

## Request processing pipeline



## Custom middleware

```
function log(request,response,next){
console.log("Logging...");
next();
}
module.exports = log;
```

## Using your own middleware

```
 const logger = require('./logger-
middleware');
```

app.use(logger);

## Helmet  (npm install express helmet)

helmet helps you secure your Express apps by setting various HTTP headers. It's not a silver bullet, but it can help!

First, run npm install helmet

```
const express = require('express')
const helmet = require('helmet')
const app = express()
app.use(helmet())
```

## Exercise

Check what morgan is and how it works