**API:**

Application programing interface or library or function which can be accessed over the internet to compute the data, get data, and post data

Example: URI (uniform resource identifier)/ address of API

<https://jsonplaceholder.typicode.com/todos>

/users

**Return type:**

JSON (java script object notation) data

Example:

{key:value}

[ {id:101,name:”nitin”,gender:”male”},

{id:101,name:nitin,gender:male},

{id:101,name:nitin,gender:male} ]

**What is RESTfull API architecture?**

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol.

A REST Server simply provides access to resources and REST client accesses and modifies the resources using HTTP protocol. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML but JSON is the most popular one.

## HTTP methods

The HTTP operations available are:

* POST (create a resource or generally provide data)
* GET (retrieve an index of resources or an individual resource)
* PUT (create or replace a resource)
* PATCH (update/modify a resource)
* DELETE (remove a resource)

Based on this information we are going to provide following RESTful APIs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.No.** | **URI** | **HTTP Method** | **POST body** | **Result** |
| 1 | Listusers | GET | Empty | Show list of all the users. |
| 2 | Users | POST | JSON String | Add details of new user. |
| 3 | /users/:userId | DELETE | JSON String | Delete an existing user. |
| 4 | :id | GET | Empty | Show details of a user. |

End point:

[https://jsonplaceholder.typicode.com](https://jsonplaceholder.typicode.com/todos)

## Resources

JSONPlaceholder comes with a set of 6 common resources:

|  |  |
| --- | --- |
| [/posts](https://jsonplaceholder.typicode.com/posts) | 100 posts |
| [/comments](https://jsonplaceholder.typicode.com/comments) | 500 comments |
| [/albums](https://jsonplaceholder.typicode.com/albums) | 100 albums |
| [/photos](https://jsonplaceholder.typicode.com/photos) | 5000 photos |
| [/todos](https://jsonplaceholder.typicode.com/todos) | 200 todos |
| [/users](https://jsonplaceholder.typicode.com/users) | 10 users |

**Note**: resources have relations. For example: **posts** have many **comments**, **albums** have many **photos**, ... see below for routes examples.

## Routes

All HTTP methods are supported.

|  |  |
| --- | --- |
| GET | [/posts](https://jsonplaceholder.typicode.com/posts) |
| GET | [/posts/1](https://jsonplaceholder.typicode.com/posts/1) |
| GET | [/posts/1/comments](https://jsonplaceholder.typicode.com/posts/1/comments) |
| GET | [/comments?postId=1](https://jsonplaceholder.typicode.com/comments?postId=1) |
| GET | [/posts?userId=1](https://jsonplaceholder.typicode.com/posts?userId=1) |
| POST | /posts |
| PUT | /posts/1 |
| PATCH | /posts/1 |
| DELETE | /posts/1 |

**Promise:**

is java script library which is developed by facebook

fetch(uri)

.then(res=>res.json())

.then(out=>console.log(out))

Resource:

Listusers

Users

/users/:userId

* [POST] endpoint/users
* [GET] endpoint/listusers (list users)
* [GET] endpoint/users/:userId (get specific user)
* [PATCH] endpoint/users/:userId (update the data for the specified user)
* [DELETE] endpoint/users/:userId (remove the specified user)

JSON.stringify(data)

JSON.parse(data)

app.get('/:id', function (req, res) {

*http://127.0.0.1:3010/100*

**CORS**

**CORS** is a node. js package for providing a Connect/Express middleware that can be used to enable **CORS** with various options.

This post shows how to enable Cross Origin Resource Sharing (**CORS**) in **Node**. **CORS** essentially means cross-domain requests. Simply using this line of code to set a header on your response will enable **CORS**

// Website you wish to allow to connect

res.setHeader('Access-Control-Allow-Origin', '\*');

// Request methods you wish to allow

res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');

// Request headers you wish to allow

res.setHeader('Access-Control-Allow-Headers', 'Access-Control-Allow-Headers, Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers,X-Access-Token,XKey,Authorization');

//methods

app. use(function(req, res, next) {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept" );

});

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs'

export class AuthService {

apiUrl = 'https://127.0.0.1:3000'

httpOptions = {

headers: new HttpHeaders({

'Content-Type': 'application/json'

})

}

constructor(private http: HttpClient) { }

login(username: String, password: String): Observable<any[]> {

return this.http.post<any[]>(this.apiUrl + '/login', {username, password}, this.httpOptions);

}

}

export class AppComponent {

title = 'app';

constructor(private authService: AuthService) {}

doLogin(f: any) {

this.authService.login(f.username, f.password)

.subscribe(

(data => {

console.log(`data ==> ${data}`);

}),

(error => {

console.log(`error ==> ${JSON.stringify(error)}`);

})

)

}

}

**Read data from request**

The query string is the part that comes after the URL path, and starts with a question mark

For example:

**?name=Flavio**

Multiple query parameters can be added using &

**?name=flavio&age=35**

Express makes it very easy by populating the Request.query object for us:

**app.get('/', (req, res) => {**

**console.log(req.query)**

**})**

If there are no query params, it’s an empty object.

This makes it easy to iterate on it using the for…in loop:

**for (const key in req.query) {**

**console.log(key, req.query[key])**

**}**

**Other Way**

app.get('/users/:userId/books/:bookId', function (req, res) {

res.send(req.params)

})

Route path: /users/:userId/books/:bookId

Request URL: http://localhost:3000/users/34/books/8989

req.params: { "userId": "34", "bookId": "8989" }

Example 2:

Python

Tweepy: library

TextBlob:

Practice:

Data type:

Number

A =55

B =6666.3433

String

A =’fkfhfhgf ‘ or a=”fhfjhgfhfgg”

Boolean

A = true or false

Object

Array : a = [111,4,6,77,”ggg”]

Json : {id:55,name:56677}

Pop() – remove last value , push() – add new value at last

Shift() – remove first value unshift() – add new value at first

Slice() – return value from given index range, splice() – remove from given index

Sort() – arrange in asc order , reverse() – arrange in desc order

emp = [{id:101,name:'nitin',gender:'male'},{id:101,name:'pooja',gender:'female'},{id:1,name:'nitin',gender:'male'},{id:201,name:'nitin',gender:'male'},{id:11,name:'nitin',gender:'male'},{id:101,name:'nitisha',gender:'female'}]

(6) [{…}, {…}, {…}, {…}, {…}, {…}]0: {id: 101, name: "nitin", gender: "male"}1: {id: 101, name: "pooja", gender: "female"}2: {id: 1, name: "nitin", gender: "male"}3: {id: 201, name: "nitin", gender: "male"}4: {id: 11, name: "nitin", gender: "male"}5: {id: 101, name: "nitisha", gender: "female"}length: 6\_\_proto\_\_: Array(0)

emp

(6) [{…}, {…}, {…}, {…}, {…}, {…}]

emp.filter(row => row.gender=='female')

(2) [{…}, {…}]0: id: 101name: "pooja"gender: "female"\_\_proto\_\_: Objectconstructor: ƒ Object()\_\_defineGetter\_\_: ƒ \_\_defineGetter\_\_()\_\_defineSetter\_\_: ƒ \_\_defineSetter\_\_()hasOwnProperty: ƒ hasOwnProperty()\_\_lookupGetter\_\_: ƒ \_\_lookupGetter\_\_()\_\_lookupSetter\_\_: ƒ \_\_lookupSetter\_\_()isPrototypeOf: ƒ isPrototypeOf()propertyIsEnumerable: ƒ propertyIsEnumerable()toString: ƒ toString()valueOf: ƒ valueOf()toLocaleString: ƒ toLocaleString()get \_\_proto\_\_: ƒ \_\_proto\_\_()set \_\_proto\_\_: ƒ \_\_proto\_\_()1: {id: 101, name: "nitisha", gender: "female"}id: 101name: "nitisha"gender: "female"\_\_proto\_\_: Objectlength: 2\_\_proto\_\_: Array(0)length: 0constructor: ƒ Array()concat: ƒ concat()copyWithin: ƒ copyWithin()fill: ƒ fill()find: ƒ find()findIndex: ƒ findIndex()lastIndexOf: ƒ lastIndexOf()pop: ƒ pop()push: ƒ push()reverse: ƒ reverse()shift: ƒ shift()unshift: ƒ unshift()slice: ƒ slice()sort: ƒ sort()splice: ƒ splice()includes: ƒ includes()indexOf: ƒ indexOf()join: ƒ join()keys: ƒ keys()entries: ƒ entries()values: ƒ values()forEach: ƒ forEach()filter: ƒ filter()flat: ƒ flat()flatMap: ƒ flatMap()map: ƒ map()every: ƒ every()some: ƒ some()reduce: ƒ reduce()reduceRight: ƒ reduceRight()toLocaleString: ƒ toLocaleString()toString: ƒ toString()Symbol(Symbol.iterator): ƒ values()Symbol(Symbol.unscopables): {copyWithin: true, entries: true, fill: true, find: true, findIndex: true, …}\_\_proto\_\_: Object

emp.filter(row => row.gender=='male')

(4) [{…}, {…}, {…}, {…}]0: {id: 101, name: "nitin", gender: "male"}id: 101name: "nitin"gender: "male"\_\_proto\_\_: Object1: {id: 1, name: "nitin", gender: "male"}id: 1name: "nitin"gender: "male"\_\_proto\_\_: Object2: {id: 201, name: "nitin", gender: "male"}id: 201name: "nitin"gender: "male"\_\_proto\_\_: Object3: {id: 11, name: "nitin", gender: "male"}id: 11name: "nitin"gender: "male"\_\_proto\_\_: Objectlength: 4\_\_proto\_\_: Array(0)length: 0constructor: ƒ Array()concat: ƒ concat()copyWithin: ƒ copyWithin()fill: ƒ fill()find: ƒ find()findIndex: ƒ findIndex()lastIndexOf: ƒ lastIndexOf()pop: ƒ pop()push: ƒ push()reverse: ƒ reverse()shift: ƒ shift()unshift: ƒ unshift()slice: ƒ slice()sort: ƒ sort()splice: ƒ splice()includes: ƒ includes()indexOf: ƒ indexOf()join: ƒ join()keys: ƒ keys()entries: ƒ entries()values: ƒ values()forEach: ƒ forEach()filter: ƒ filter()flat: ƒ flat()flatMap: ƒ flatMap()map: ƒ map()every: ƒ every()some: ƒ some()reduce: ƒ reduce()reduceRight: ƒ reduceRight()toLocaleString: ƒ toLocaleString()toString: ƒ toString()Symbol(Symbol.iterator): ƒ values()Symbol(Symbol.unscopables): {copyWithin: true, entries: true, fill: true, find: true, findIndex: true, …}\_\_proto\_\_: Object

emp.filter(row => row.id >200)

[{…}]