REST API

API → it is a way of communication between two systems or two environments; two isolated systems. A good example for this would be, let us take a seller. He has a profile in Amazon which is a middleware. Amazon provides a platform to list your products so that customers who come to amazon would buy it. if your product is selected, then Amazon would ask you to send the product to a particular address. In this context Amazon is an API.

In our context, without using express library, we can create a server. What express does, is the library itself takes care of going into the node environment and creates a server for you. using listen(), the express server is creating server behind the scenes. The express is giving an API of listen() function to communicate with NodeJS indirectly which is more convenient for me. The idea is always similar only context changes.

Even in the browser script, the document object is basically an API so that we can communicate our information by defining certain functions in our browser script which tells document object what is going to happen, and the document object manipulates the DOM. We are using document and not directly doing something to change.

This addEventListener is an API defined on the document object which we are calling so that whenever something happens in the DOM, that notification goes into the function, the function that we have defined as the second parameter. We are not concerned about the logic behind. APIs are ultimately everywhere. It is just that context changes.

REST:

Representational State Transfer.

Rest API is a protocol or an architectural concept in order to avail their application data. They are making it available to us. When they say, API is REST compatible we know how to put in the details in it.

REST API is following a particular structure to create your API

REST API is specific to URL. If we want to specify a particular structure to call our URLs and get the data. First, we will create an API and then we will make it RESTFUL.

We have to create an API so that the browser and browser side script would be able to communicate with certain route to send certain Data, that will create an API. There is a specific pattern if we follow that will create a RESTFUL API.

In the **`user.json`** file, we have defined properties. We have given numbers inside the key name and it is another nested object itself. We have given unique numbers to the users. This entire structure for now we will mock as database. We have to create an API such that, in my URL, when we type localhost:3000/anything we will get the data accordingly.

First, we want to get specific user object based on an ID. Based on that my server will go through it and get an object. For that we can create app.get and provide API route in it.

```
app.get('/getUserFromDB', (req, res) => {
    res.send('ASD')
})
```

Let's us test it out to see if it is working or not.

Now, let us specify a user id between 1 & 10 through the URL so that the backend will know which user it needs to fetch. We can do it by using query parameter or by giving the path.

We can write **`userid=123`**. So, this user id should be in the range of 1 to 10. If I'm providing a **`userid`** between the range of 1 to 10. Based on this value, when we pass **`userid=1`**, we should the details of first user in the **`user.json`** file. So, we want to accept the parameter and fetch the object accordingly.

We will user **`req.query`** and not use **`req.params`** or **`req.body`** because we are putting in a query parameter. We could have used **`req.params`** if we have put **`'/getUserFromDB/:id'`**.

Let us for now console.log(**`req.query`**) and then put **`res.send`**, because if we don't put this, the browser will keep on loading as we have not sent anything back from the server and browser waiting for the response will keep on loading. User can still scroll to your content, but it just a way of specifying to the user that some communication is going on with the server and is waiting for a response from server.

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

app.get('/getUserFromDB', (req, res) => {
    console.log(res.query)
    res.send('Over')
})

app.listen(3000, () => console.log('Server Started'))
```

Now we can see in the terminal, for the **`console.log(res.query)`** is getting logged as **`{userid: '1'}`**. If we change the URL to **`localhost:3000/getUserFromDB?id:1`**, then in the terminal we can see it mentioned as **`{id: '1'}`**.

All we need to do is since we have the id, we need to cross reference with the object and get the object in the function itself and put it inside **`res.send`** so that it would display on the browser.

Objects and functions we can do, but it does not create a closure. If we are using variables and functions with global scope, then it would be 70% correct. But if we properly define any function along with its lexical scope is a closure. Lexical scope is everything defined along with the function outside it.

We have to get the user based on the id. We are expecting a **`userid`** specifically. The **`userid`** is the query name and the value to be the number or id of the user. We need to reference json in our express. Since it is json we can directly call a require() which is an API that it understands if it is a module or file or json. We can de-structure to get the id. So,

```
const user = require('./db/users.json')
const express = require('express');
const app = express();

app.get('/getUserFromDB', (req, res) => {
    console.log(res.query)
    const {userid} = req.query
})

app.listen(3000, () => console.log('Server Started'))
```

If we need to access the first user, then we can say **`user`** [1]. It put in square brackets because we cannot have directly number. Basically, our property name cannot have number as a starting character, json or javascript object notation rule. It would get

confused. So we cannot use user.1 and so we can do **user**[], just like we do it in arrays. So, we will put in **user**[1] for first user and till 10 respectively.

Now we want to make it dynamic, so the value 1 to 10 would be in **userID**.

```
const user = require('./db/users.json')
const express = require('express');
const app = express();

app.get('/getUserFromDB', (req, res) => {
    console.log(res.query)
    const {userid} = req.query
    const foundUser = user[userID]

    console.log(foundUser)

    res.send('Over')
})

app.listen(3000, () => console.log('Server Started'))
```

Change the URL to, **localhost:3000/getuserfromdb?Userid=2**

We would get “Over” in the browser, but in the terminal, we can see the details of the user whose userID we have mentioned in the URL. We can do it by using the send function as well. So, we can do res.send(foundUser) and we can see the object in the browser.

```
const user = require('./db/users.json')
const express = require('express');
const app = express();

app.get('/getUserFromDB', (req, res) => {
    console.log(res.query)
    const {userid} = req.query
    const foundUser = user[userID]

    console.log(foundUser)

    res.send(foundUser)
})

app.listen(3000, () => console.log('Server Started'))
```

As a good approach always first solve…

We can use try…catch also, it should throw an error, but the value is getting saved as undefined, which is not an error. So, it is good to use the if-block, if the **`userID`** actually exist in the **`req.query`** and then call it. Always try to make it work and then think of different conditions that might create an error in the approach.

Coming to the REST-wala part,

We have defined that we need to **`getUserFromDB`** and then put **`urserid`** and the id number. If we change /**`getFromUserID`** to /**`dontGetUser`** and still we will be getting the value. This route will give user object irrespective of the name you give to it.

Even if we give **`app.noServer`** instead of **`app.listen`** will it make sense to anyone who is using our api? We need to get all the details in the documentation. Since this is GET, we can also make it POST or delete.

Rest standardized. For example we have users or resource or collection, if we define a get route which sought of emulates, user/(particular id), then we can say that particular route is restful route. if I put or write my route in such a way that it is a post route, then we have users or collection name, then we can say this route is restful in nature. Till now whatever we have created is simple API, but if we follow this particular standard, to get a particular user based on the id.

All we need to know if the name of the collection like the jsonplaceholder typecode itself. It follows the exact pattern. This should give us list of all users. An array of entire users in the data base. If we want to access a specific user with their ID I wont have to put query parameters. That would be non-restful way.

However, if we give the /post, it is a restful route because when we enter /post/1 we get the data of the user whose id is 1.

Our collection name is users, so now we will name our path as /users. This signifies that we want to do some operations regarding users in our db. The idea is to get userwith specific id and for that we are using query parameters. But restful api tells us that we should not use it rather use a structure which would say the id as the second path. So, **`/users/:userId`**

Instead of **`req.query`**, it would be in **`req.params`**. We will not be putting question mark in the url. If everything is working properly if we give a question mark in the url, then it should not give any details. If we have to get a user with **`userid`**: 2 then we have to **`write localhost:3000/users/2`**.

```
const user = require('./db/users.json')
const express = require('express');
const app = express();

//users
//user based on ID
app.get('/users/:userId', (req, res) => {
    console.log(req.params)
    const {userId} = req.params

    const foundUser = users[userId]
    res.json(foundUser)
})

app.listen(3000, () => console.log('Server Started'))
```

Whatever is the name of the .json file, should be given in the route path.

Storage devices only stores data. When we are actually running something, the OS loads the application in the RAM and then using RAM and processor, that application is run.

NodeJS is a single threaded language. Just as an engine is not directly moving the vehicle, but it provides power to the tyres to move. In the same way threads are simple computational unit which software can use. NodeJS occupies one single thread and keeps on running on it. and whatever code is supposed to run asynchronously, NodeJS off-loads it asynchronously to other threads.

We have actually defined the second route for now. Let us go with the first one. We have make it in such a way that it should not accept any parameter and it should give me list of all the objects. So it represents one single route that will give us all the data.

Now the structure of json object is in key:value pair and not an array. We need to create this object as an array and then send it back. We have to say,

```
//users
app.get('/users', (req, res) => {
    const keys = Object.keys(users) // [1,2,3,4]
    const userArray = keys.map(id => users[id])
    res.json(userArray)
})
```

What we have defined here is when we pass in a user id then we would get the specified user data. And if we don’t then we would get an array of all the then users.

If we want to update a specific id, we can follow the same pattern; logic will be different, but we can only make changes to one userId at once. Again finally, if we want to delete then my pattern would be same except it would be the same.