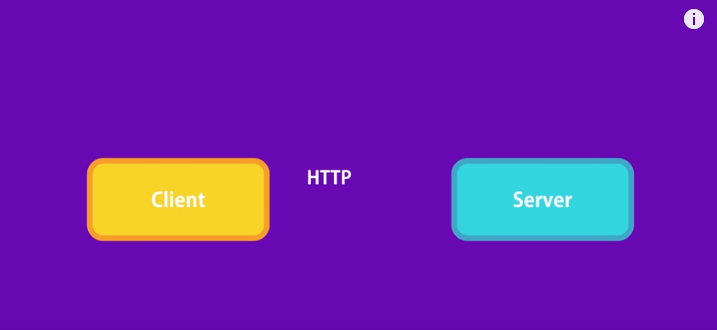
**BUILD RESTful APIs/ RESTful Services with Node and Express (16.02.2019)**

* First let’s talk about the REST (Representational State Transfer)

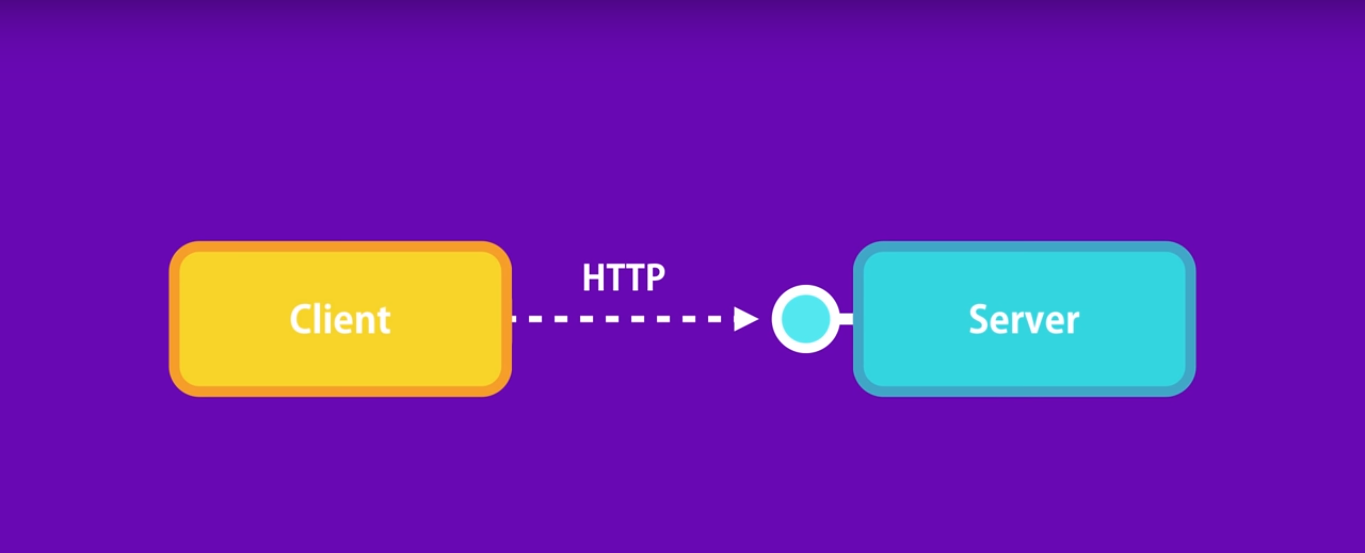


All application these days follows this structure

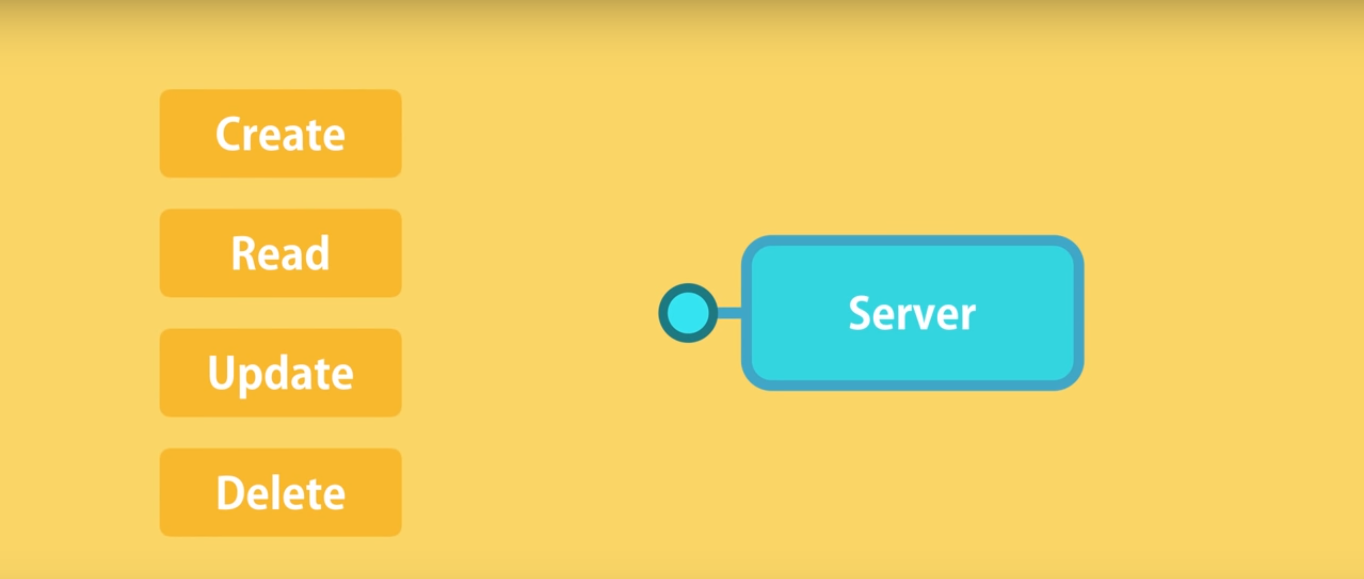
App = App itself is Client / Front end part

Server = Under the hood app needs to talk to Server / Backend to get or save the data

This communication happens with the help of HTTP Protocol, the same protocol that powers the web



* On the server there are the bunch of services that are accessible through HTTP Protocol. The client can directly call this services by sending HTTP request.
* This is where REST comes into the picture. **REST = Representational Transfer**
* We will be doing CRUD Operations using REST



* Lets talk about the <http://vidly.com/api/customers>

https:// = protocol

vidly.com = domain

api = basic convention to use term api somewhere in the url

customers = resources (This is the end point)

* Every http request has the verb/method that determines is type or intention

GET = For getting data

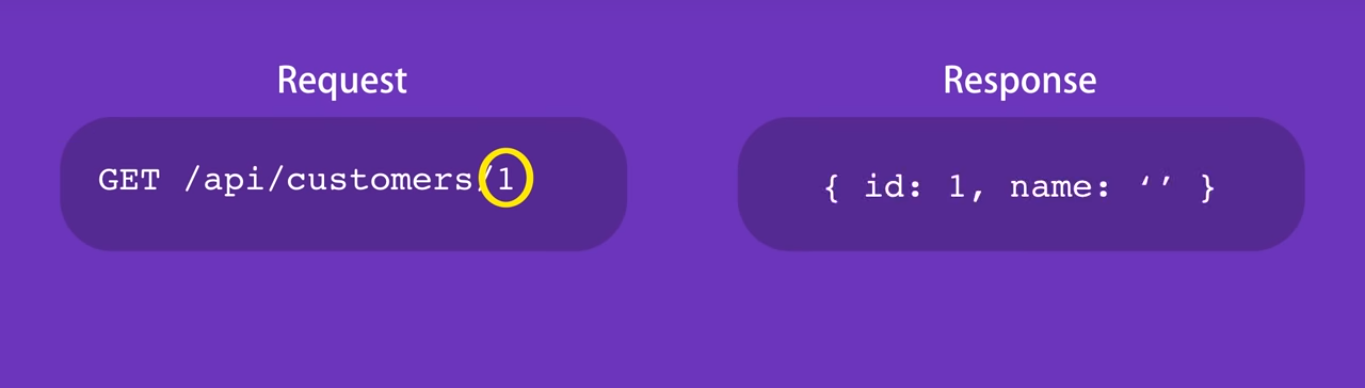
POST = For creating data

PUT = For updating data

DELETE = For deleting data

1. GET

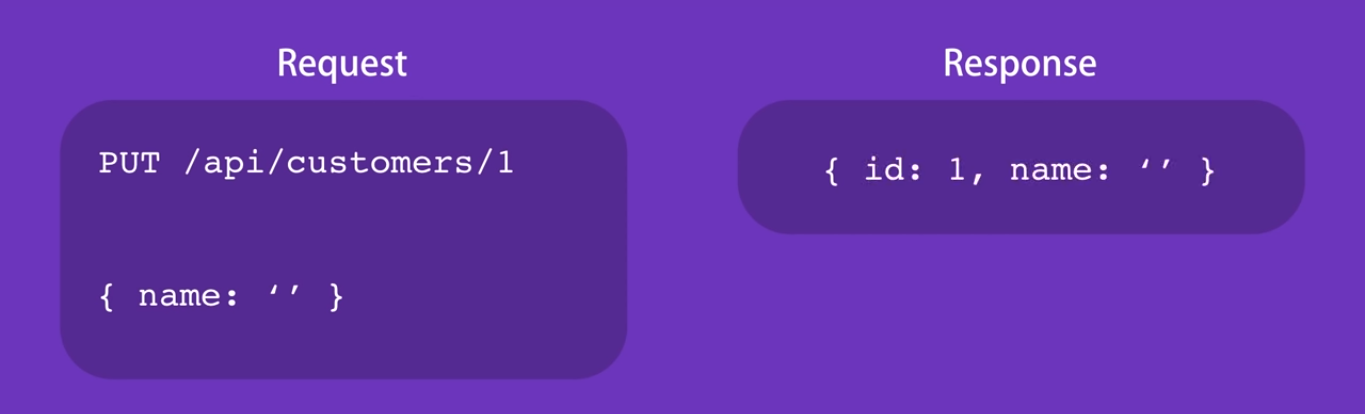




POST:

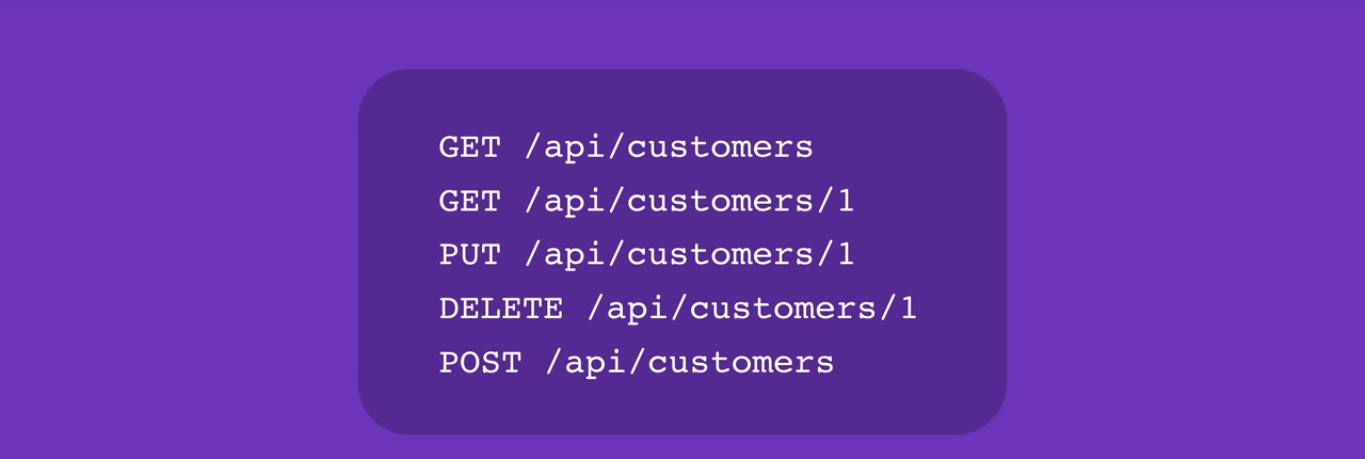


PUT (for update):

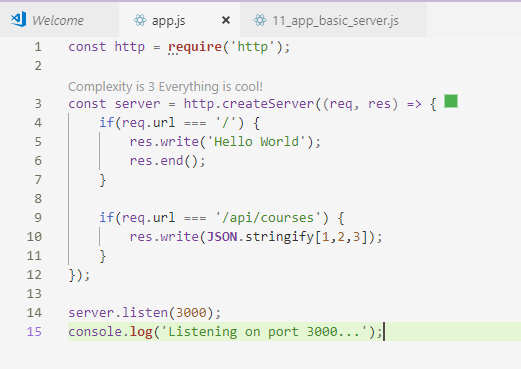


DELETE:





* So throughout the section we will learn how to use the express framework to build the RESTful service for managing the list of customers (Without database – we will be using array)



* This is the basic server created in node js. This is good, but when there are lots of routes and if conditions within the server.
* Here framework comes into picture. A framework gives the application proper structure, so we can add as many routes while keeping our code maintainable. Most popular framework is **express**. It is fast, light weight and perfectly documented
* Go to npm and search epress:

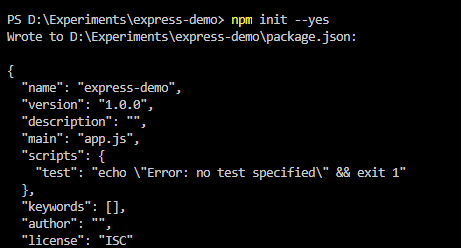
<https://www.npmjs.com/package/express>

**Installation:**

Open the terminal in new project folder and

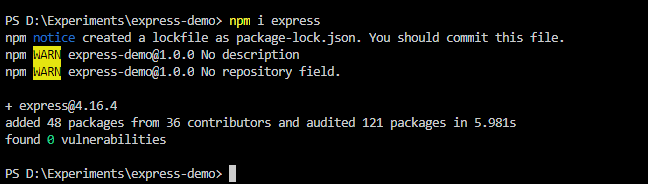
Type: Run below commands

**npm init –yes**

****

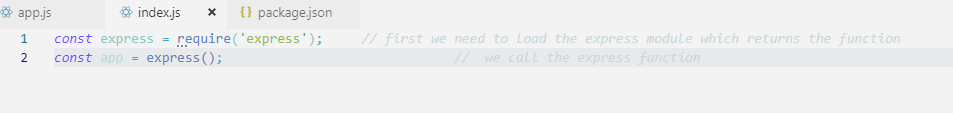
Now we have package.json file created in project

**npm i express**

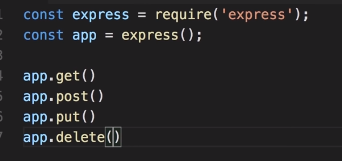


**CREATE THE EXPRESS WEB SERVER**

a. Create the index.js file

First we load the express module and call the express function

Now this express method have the bunch of useful methods like



We discussed this earlier about requests.

Lets start with get request:

*const* express = require('express'); *// first we need to load the express module which returns the function*

*const* app = express(); *// we call the express function*

*// 1st parameter is path '/' is for home, and 2nd para is callback function*

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

res.send('Hello World');

});

*// 1st parameter = port, 2nd parameter (optional) = displaying message*

app.listen(3000, () *=>* console.log('Listening in port 3000 ...'));

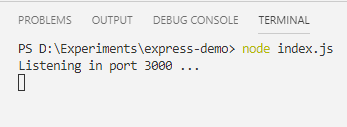
app.get contains 2 parameter:

**1st = route/path in url, 2nd = callback function with message**

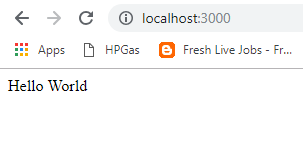
app.listen contains 2 parameters:

**1st = port number, 2nd (optional)= message**

Lets run in terminal:



In Browser:



Lets create another request

In /api/courses mainly **deals with databases, we will deal will array to understand express**

*const* express = require('express'); *// first we need to load the express module which returns the function*

*const* app = express(); *// we call the express function*

*// 1st parameter is path '/' is for home, and 2nd para is callback function*

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

res.send('Hello World');

});

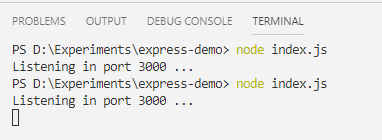
*// another get request*

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

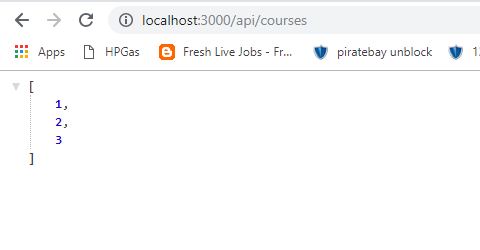
res.send([1,2,3]);

});

Stop and restart the server in terminal:



Browser:

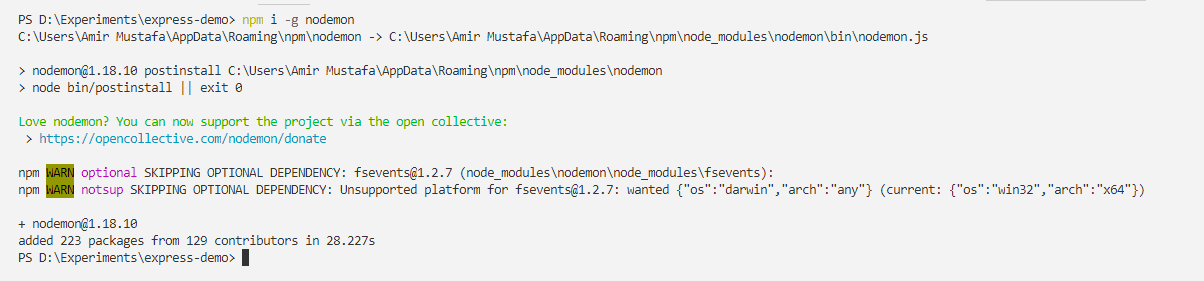


So array comes here. We can move all the routes related with /api/courses to a separate js page like courses.js

**Nodemon**

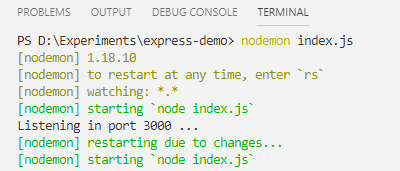
* So far we have seen every time we have to make change in the code we have to stop the server in the terminal and restart it which is a very tedious way.
* For this we have to install the js package called **nodemon** shortenedfor **Node Monitor** whichworks to stop the package.

**npm i -g nodemon //g is for global installation, i is for install**



* Instead of **node index.js**, we will write **nodemon index.js** in terminal

Suppose you make some changes in code eg. string in terminal nodejs will be automatically be written



Though you need refresh the browser, but no need to restart the terminal server.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Environment Variable**

* In real time giving port 3000 does not work in production(server) which is dynamically assigned. For test we can use any port of our choice
* For this environment variable is used.

*// Id environment port is there take that othewise default*

*const* port = process.env.PORT || 3000;

*// 1st parameter = port, 2nd parameter (optional) = displaying message*

app.listen(port, () *=>* console.log(`Listening in port ${port} ...`));

Instead of static port, we have taken in a constant and of environment port is present take that other wise second constant

* We have also used back ticks in message to show port it is using. At present will show 3000 as nothing .

SET ENVIRONMENT VARIABLE: (Try running from command prompt/terminal and not from editor terminal)

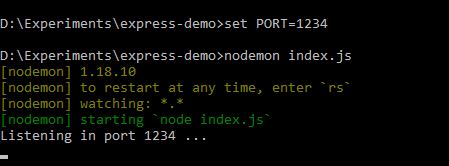
In MAC/LINUX:

**export PORT=1234**

In Windows:

**set PORT=1234**

**Now run nodemon index.js**

****

**Route Parameters:**

* Through route params we get single set of record by passing id in the request

*// route parameter - getting single set of record based on the id eg, /api/courses/1*

*// 1st parameter = port, 2nd parameter (optional) = displaying message*

app.get('/api/courses/:id', (req, res) *=>* {

res.send(req.params.id);

});

app.listen(port, () *=>* console.log(`Listening in port ${port} ...`));

Multiple parameters:

We can pass as many no. of paramaters eg if we pass date and year

*// multiple params*

app.get('/api/posts/:year/:date', (req, res) *=>* {

res.send(req.params); *// parameter object*

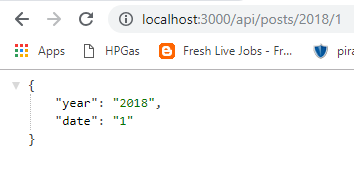
// res.send(req.params.year); *// year*

// res.send(req.params.date); *// date*

});

app.listen(port, () *=>* console.log(`Listening in port ${port} ...`));

Browser: (request params object)



Query String Parameters

* What ever we give after question marks is query parameters

Eg.

In url: <http://localhost:3000/api/posts/2018/1?sortBy=name>

Yellow = route params

Green = query string params

Eg 2: <http://localhost:3000/api/posts/2018/1?sortBy=name&code=5>

*// query string params*

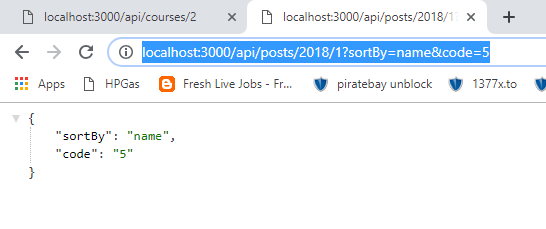
app.get('/api/posts/:year/:date', (req, res) *=>* {

res.send(req.query); // Here we write query instead of params

});

app.listen(port, () *=>* console.log(`Listening in port ${port} ...`));

Browser:



Query Parameters = Used for additional data for the backend services /optional values

Route Parameters = Used for essential or required values

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Handeling GET Requests**

* Here we will take list of courses and will find will try to find specific course.
* If course is found return that course otherwise show message of course not found

EG: index.js

*const* express = require('express'); *// first we need to load the express module which returns the function*

*const* app = express(); *// we call the express function*

*// array of courses*

*const* courses = [

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

{id: 2, name: 'course 2'},

{id: 3, name: 'course 3'},

];

*// Id environment port is there take that othewise default*

*const* port = process.env.PORT || 3000;

*// route parameter - getting single set of record based on the id eg, /api/courses/1*

*// 1st parameter = port, 2nd parameter (optional) = displaying message*

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

res.send(courses);

});

*// find specific course*

app.get('/api/courses/:id', (req, res) *=>* {

*// here we will write some logic to get the specific id data*

*const* course = courses.find(c *=>* c.id === parseInt(req.params.id));

*// if we do not find the course in array*

*if*(!course) res.status(404).send('The course with the given ID was not found');

*// othyerwise*

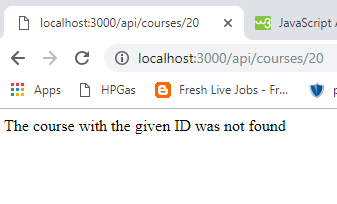
res.send(course);

});

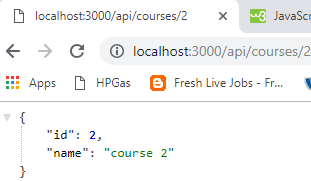
app.listen(port, () *=>* console.log(`Listening in port ${port} ...`));

Browser:

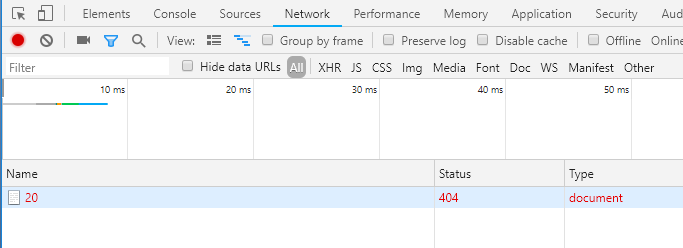
1. Unknown id



1. Id present in course array:



You can also check in the dev tools:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Handeling POST Requests**

* Here we will see how POST Request is sent

Eg. here we are using array: therefore length+1, in real time data will come from db.

Here we are assuming in the request body there is object and and that object has the name property.

Eg. name: req.body.name

In order to use this we have **to enable parsing of JSON objects** in the body of the requests

*// Handeling POST requests*

app.post('/api/courses', (req, res) *=>* {

*const* course = {

id: courses.length + 1, // because new id will be last element no. + 1

name: req.body.name // reading body for this we will use body first in top

};

});

* Adding express.json() in top

express.json() is adding a piece of middleware. We use this middleware in the request processing pipeline. We will explore this more later in the section.

*const* express = require('express'); *// first we need to load the express module which returns the function*

*const* app = express(); *// we call the express function*

app.use(express.json()); *// Way to use in body*

* Finally we will push the new course in the courses array.

*// Handeling POST requests*

app.post('/api/courses', (req, res) *=>* {

*const* course = {

id: courses.length + 1,

name: req.body.name

};

*// Pushing the new course in the course array*

courses.push(course);

*// receiving the response*

res.send(course);

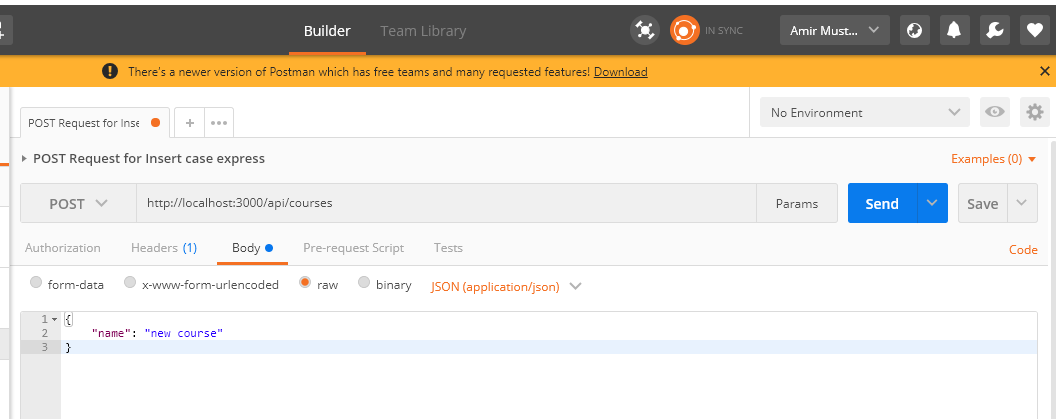
});

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

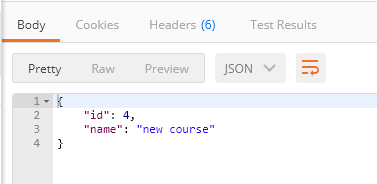
**Postman:**

* Download chrome Postman extension and make a new collection related with the project.
* Now run the above code in postman, POST request and in body(select raw and json/application from other dropdown instead of text)

**Request:**



**Response:**



* So now when you hit get request you will get 4 data (including the new data inserted from POST

)



So this is how we test HTTP services in postman

* For this we have assumed there in name parameter in body. If we forget to put name or misspelled that’s where **input validation** comes into the picture

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Input Validation**

* We should always validate whatever is sent through client(i.e. from Postman). If not found or required conditions does not match will throw **400 i.e. Bad Request.**
* **return** isfor stoppingremaining code

Eg.

app.post('/api/courses', (req, res) => {

*// validting the request*

if(!req.body.name) { // if body of request does not contain name

*// 400 Bad Request*

res.status(400).send('Name is required');

return;

}

if(req.body.name.length < 3) { // if body of request contains value of name < 3

*// 400 Bad Request*

res.status(400).send('Name should be minimum 3 characters');

return;

}

*// if(!req.body.name || req.body.name.length < 3) { // can both vaidation be in one*

const course = {

id: courses.length + 1,

name: req.body.name

};

*// Pushing the new course in the course array*

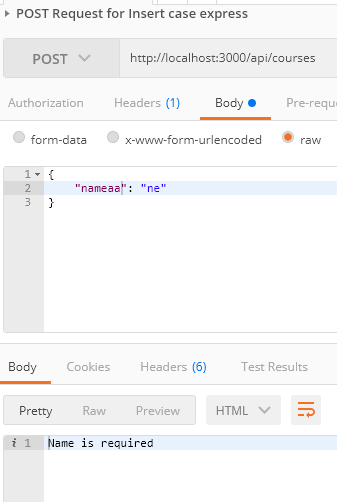
courses.push(course);

*// receiving the response*

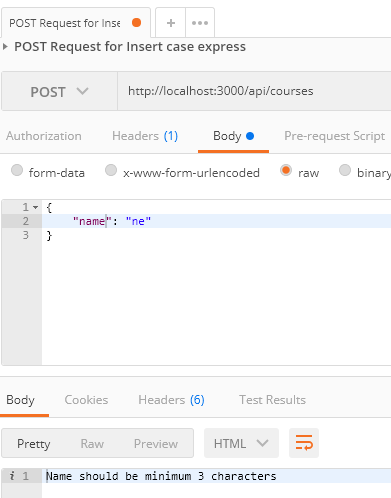
res.send(course);

});

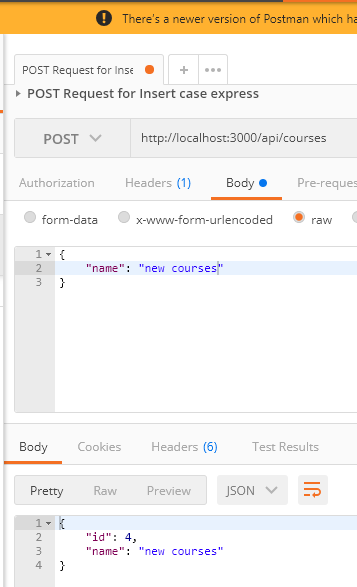
Name validation:



Length validation:



Ok condition:



* In real time validation will be complex. For this there is a npm package manager called joi

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**JOI**

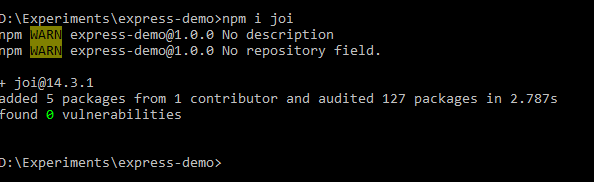
* Joi is the input validation npm package manager

<https://www.npmjs.com/package/joi>

* Install joi in project

npm i joi

For getting **specific version**: npm I [joi@13.1.0](mailto:joi@13.1.0)



* We will first include Joi in top with J capital in

const Joi = **require**('joi'); *// for input validation*

const express = **require**('express'); *// first we need to load the express module which returns the function*

const app = express(); *// we call the express function*

app.use(express.json()); *// Way to use in body*

* With joi first we need to define a **schema**.

A **schema** defines the shape of our objects. What properties do we have in that object. What is the type of each property. Do we have an email? Do we have the string.? Do we have min or max no. of characters? Do we have the numbers? What range should that number be?

* So this is the job of the schema

app.post('/api/courses', (req, res) => {

// schema

const schema = {

name: Joi.string().min(3).required()

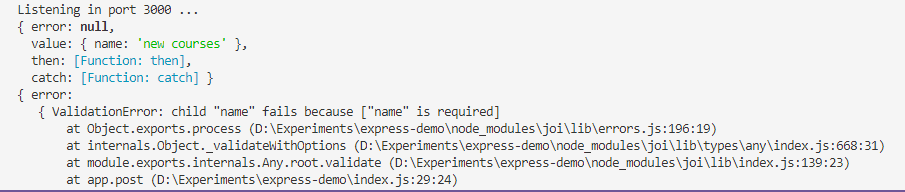
};

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

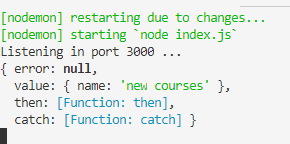
console.**log**(result);

)};

If we send wrong validation like name not mentioned: will throw error in terminal



If Ok



* So we will write like this:

app.post('/api/courses', (req, res) => {

const schema = {

name: Joi.string().min(3).required()

};

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

*// validting the request*

if(result.error) {

*// 400 Bad Request*

*// res.status(400).send(result.error); // complex json*

res.status(400).send(result.error.details[0].message); // specific msg

return;

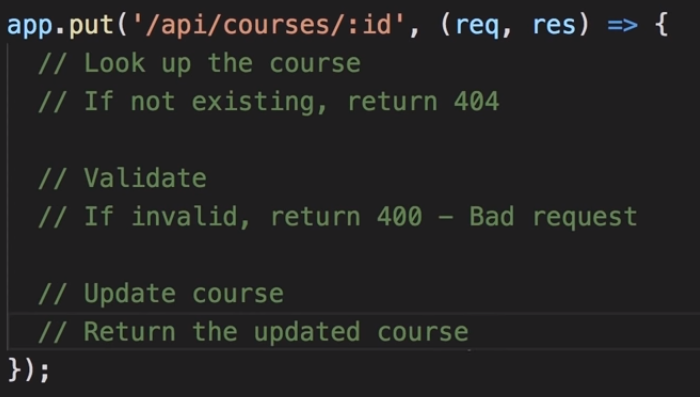
}

});

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Handeling PUT Request (Update the data)**

🡪Here we will see how to update a course. We will handle following course as commented step by step



* We have to update specific id therefore params (i.e. :id) is required

*// Handeling PUT requests*

app.put('/api/courses/:id', (req, res) => {

*// Look up the course*

const course = courses.find(c => c.id === **parseInt**(req.params.id));

*// If not existing, return 404*

if(!course) res.status(404).send('The course with the given ID was not found');

*// Validate*

*// this function can be reused if we write in a function*

*// using object destructured method to target property instead of result.error, i.e. accessing error property*

const { error } = courseValidate(req.body); *// equivalent to result.error*

if(error) {

res.status(400).send(error.details[0].message);

return;

}

*/\* const result = courseValidate(req.body);*

*// If invalid, return 400 - Bad request*

*if(result.error) {*

*res.status(400).send(result.error.details[0].message);*

*return;*

*} \*/*

*// Update course*

course.name = req.body.name;

*// Return the updated course*

res.send(course);

});

*// function handeling the validation*

function **courseValidate**(course) {

const schema = {

name: Joi.string().min(3).required()

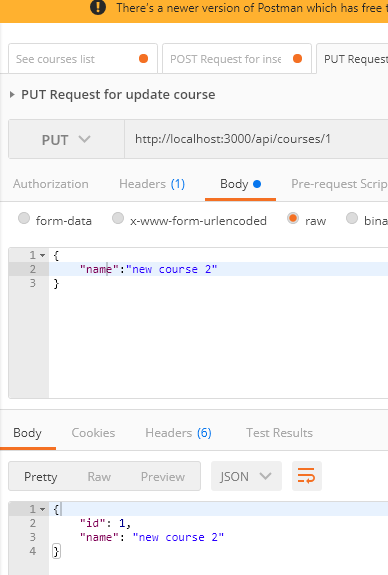
};

return Joi.validate(course, schema);

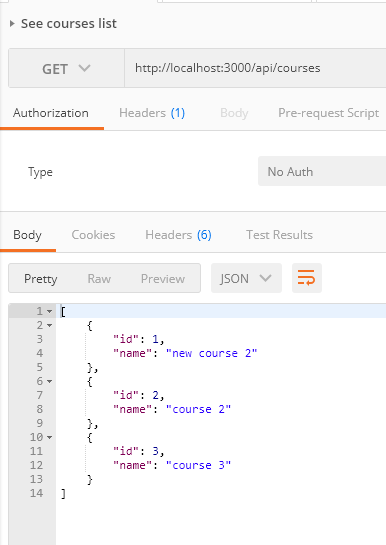
}

app.listen(port, () => console.**log**(`Listening in port ${port} ...`));

POSTMAN



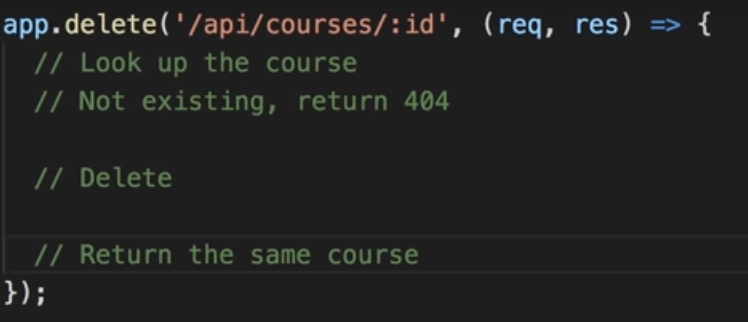
We see the course with ID 1 is updated from **course 1** to **new course 2**after clearing all validations:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Handeling Delete Requests**

* We need to delete specific id there params is requires (i.e. :id)



* To delete an element from array special method splice is used

Eg1. delete element

let arr = ["I", "study", "JavaScript"];

arr.splice(1, 1); // from index 1 remove 1 element

alert( arr ); // ["I", "JavaScript"]

Eg2. In the next example we remove 3 elements and replace them with the other two:

let arr = ["I", "study", "JavaScript", "right", "now"];

// remove 3 first elements and replace them with another

arr.splice(0, 3, "Let's", "dance");

alert( arr ) // now ["Let's", "dance", "right", "now"]

* Lets see it in program:

*// Handeling Delete request*

app.delete('/api/courses/:id', (req, res) => {

*// Look up the course*

const course = courses.find(c => c.id === **parseInt**(req.params.id));

*// Not existing, return 404*

if(!course) res.status(404).send('The course with the given ID was not found');

*// Delete - in order to delete we will take out index of that course that needs to be deleted*

const index = courses.indexOf(course);

courses.splice(index, 1);

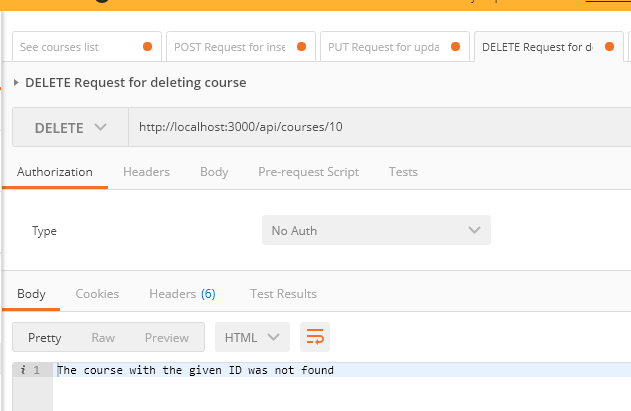
*// Return the same course*

res.send(course);

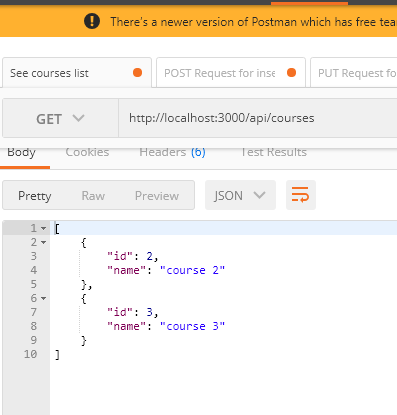
});

Postman:

Delete request



Course with id 1 removed:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ref - <https://www.youtube.com/watch?v=pKd0Rpw7O48&feature=youtu.be>