Date : 21/05/2018

**To Install Express Js -**

To install the Express Js you need to install first Node Js.

**First Method –**

Open Your command Prompt(Windows/ Node )

Make a App directory.

Go into current directory.

Now create a “package.json” file using command “**npm init**”

It will ask to fill something

Once you created then run anoher command

“npm install express --save”

Create an index.js file

Now you are ready to go

**To run the application use the command –**

**“node index.js”**

**Second Method** –

Using Express Cli

You need to install express application generator

Use command “ npm install express –generator –g”

Display the command option using “express –h”

It would show all the help section to you

Now create an Express app named as “myapp”

The list of command is below

$ express --view=pug myapp

create : myapp

create : myapp/package.json

create : myapp/app.js

create : myapp/public

create : myapp/public/javascripts

create : myapp/public/images

create : myapp/routes

create : myapp/routes/index.js

create : myapp/routes/users.js

create : myapp/public/stylesheets

create : myapp/public/stylesheets/style.css

create : myapp/views

create : myapp/views/index.pug

create : myapp/views/layout.pug

create : myapp/views/error.pug

create : myapp/bin

create : myapp/bin/www

Then install the dependencies –

$ cd myapp

$ npm install

To debug the app use this command –

> set DEBUG=myapp:\* & npm start

Now you are ready to go check this command in your browser-

<http://localhost:3000/>

The generator app has the following directory structure.

.

├── app.js

├── bin

│ └── www

├── package.json

├── public

│ ├── images

│ ├── javascripts

│ └── stylesheets

│ └── style.css

├── routes

│ ├── index.js

│ └── users.js

└── views

├── error.pug

├── index.pug

└── layout.pug

7 directories, 9 files

To create first application in Express -

Create Index/app.js file what ever you want

Create a server The syntax is like –

First load the express module –

Const express = require(‘express’);

Cost app = express(); ---------this represent the our application

Now this app object has bunch of useful method to perform the operation

Like –

App.get();

App.post();

App.put();

App.delete();

All this method represent the http request .

App.get()

This method takes two arguments. The first argument is “url” and second argument is the call back function means it is going to call back when we have the http request.

The code is –

// to create the server

const express = require('express');

const app = express();

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

res.send("This is my First Application");

});

// '/' represent the root url

// to listen the port on the server

app.listen(3000, () => console.log("Listening port on 3000"));

now run the application

use the command “node index.js”

Goto your browser – “localhost:3000”

NOTE:

The req object has bunch of the useful properties look for the documentation to go more deeper.

Expressjs.com>apirefrence>version 4.0>request>all method

To assign the port no dynamically -

The Syntax is –

// to assign the port no dynamically, because when you will host the application your port no

// will assign dynamically so you need to change here

// express js gives the environment variable called port to assign the port no dynamically.

// PORT

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

/\*....

NOTE- here process is the global object and it has the property called env(environment)

and after that we are adding here name of he environment variable which is PORT

if it is set then port no will assign automatically if it is not

then port no will assign 3000.

Now make some changes in app.listen method(port, () => console.log("Listening port on ${port}"));

instead of "" in console.log("") use single back tag (`........`);

....\*/

// to listen the port on the server

app.listen(port, () => console.log(`listening port on ${port}`));

To fetch the data from Url -

The syntax is –

App.get(‘/api/courses’ ,(req, res) => {

Res.send([1,2,3,4,5]};

});

The code is –

// to fetch the data from url in array of number format

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

res.send([1,2,3,4,5]);

})

Run the command

<http://localhost:3000/api/courses>

// to fetch the data from url in array of string format

app.get('/course\_name', (req,res) => {

res.send([

"Math",

"Science",

"Geography",

"Computer"

]);

})

Run the command –

<http://localhost:3000/course_name>

TO RUN THE SERVER AUTOMATICALLY INSTALL ONE NODE PACKAGE MANAGER --

“npm install –g nodemon” or

“npm I –g nodemon”

Now to deploy the application run the command

**“nodemon index.js”**

To set the Port environment variable through the command line-

Set PORT=5000

Now run the application

You will get the message in console that your application is running in different port no which you have assigned now.

C:\Users\Wish\Desktop\exp\_app>set PORT=5000

C:\Users\Wish\Desktop\exp\_app>nodemon index.js

[nodemon] 1.17.4

[nodemon] to restart at any time, enter `rs`

[nodemon] watching: \*.\*

[nodemon] starting `node index.js`

listening port on 5000

Route Parameters -

// to fetch the data from url in array of number format

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

res.send([1,2,3,4,5]);

})

/\*... Route Parameters

in above example we set the url '/api/courses' for the no of courses

so what if my endpoint is like this '/api/courses/a' to implement this

idea see the below expample -

...\*/

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

// now request to listen the particular id we need to request the params.id

res.send(req.params.id);

});

Run the command in browser –

<http://localhost:5000/api/courses/4>

Output will get – 4

Another example –

// Anohter example of route parameter

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

res.send([

"Monika",

"harika",

"Sankar",

"Aakash",

"Khagesh"

])

});

// to create route parameters for the friends

app.get('/friends/:name', (req,res) =>{

res.send(req.params.name);

})

Run on the browser –

<http://localhost:5000/friends/skymapglobal>

Output :skymapglobal

**Route multiple parameters –**

// set the multiple parameter in your route to bulid a complex application

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

res.send(req.params);

})

Run on the browser –

<http://localhost:5000/api/posts/2018/4>

Output: {"year":"2018","month":"4"}

**Query String Parameter –**

It is used to provide additional data for backend services.

Example-

// set the multiple parameter in your route to bulid a complex application

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

res.send(req.query);

})

NOTE:

Instead of send the “params” you need to send the “query” query parameter is stored in the object which have the key value pairs.

Run on the browser –

<http://localhost:5000/api/posts/2018/4?sortBY=name>

Output - {"sortBY":"name"}

Handling the GET Requests- (how to respond the http Get request )

// handling the Get Request Method so first define the

// to define the array of computer courses

const computer = [

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

{ id: 2, name: 'Angular JS'},

{ id: 3, name: 'Postgress Sql'},

];

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

res.send(computer);

});

app.get('/computer/:id' , (req, res) =>{

// emplement some logic

const cs\_course = computer.find( c => c.id === parseInt(req.params.id));

if(!cs\_course) res.status(404).send('The course with given ID is not found');

res.send(cs\_course);

})

Run on the browser –

<http://localhost:5000/computer/3>

Output: {"id":3,"name":"Postgress Sql"}

<http://localhost:5000/computer/4>

Output: The course with given ID is not found

// to define the array of computer courses

const computer = [

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

{ id: 2, name: 'Angular JS'},

{ id: 3, name: 'Postgress Sql'},

];

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

res.send(computer);

});

app.get('/computer/:id' , (req, res) =>{

// emplement some logic

const cs\_course = computer.find( c => c.id === parseInt(req.params.id));

if(!cs\_course){

res.status(404).send('The course with given ID is not found');

}

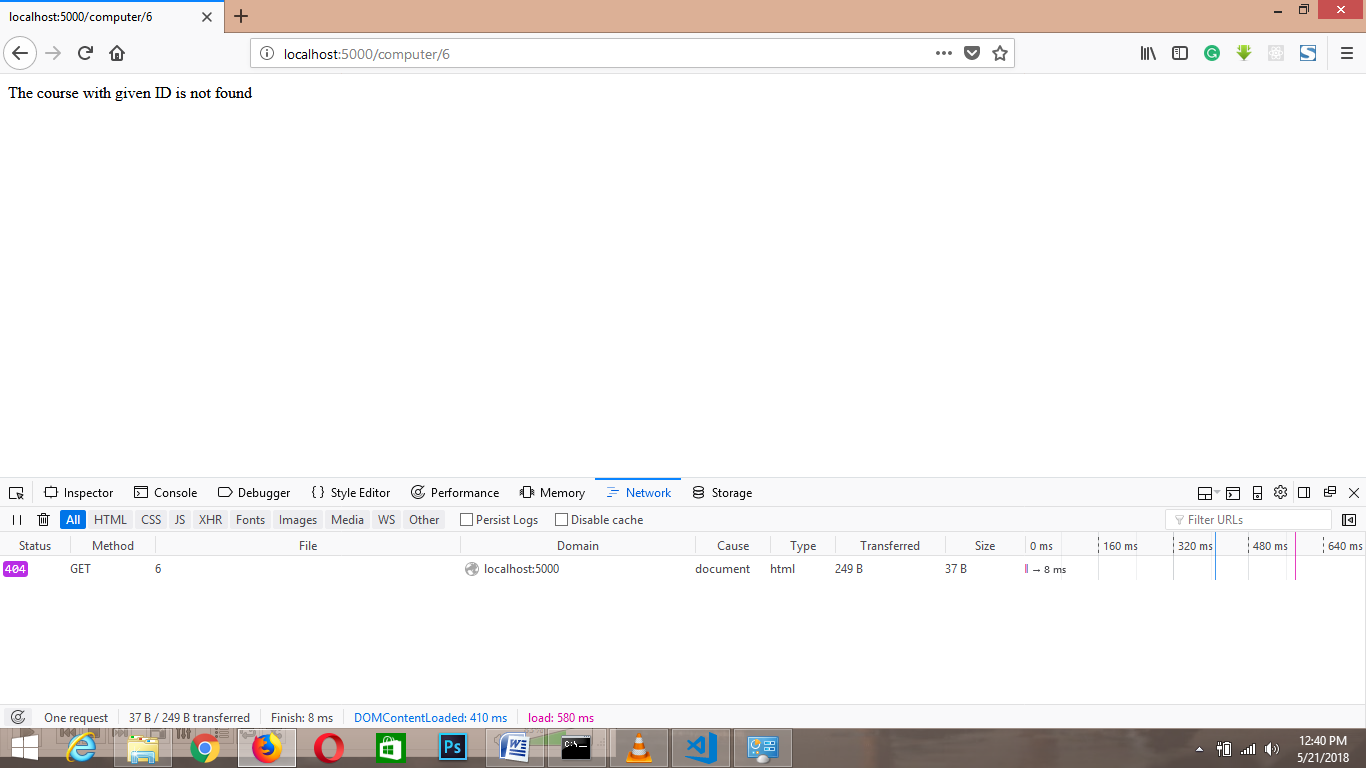
else{

res.send(cs\_course);

}

})

To check the 404 errors go to developer tool or inspect elements > go to Network tab.



Handling the POST Request – (how to respond the http post request )

const express = require('express');

const app = express();

app.use(express.json());

// handling the Get Request Method so first define the

// to define the array of computer courses

const courses = [

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

{ id: 2, name: 'Angular JS'},

{ id: 3, name: 'Postgress Sql'},

];

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

res.send(courses);

});

// to handle the http post request method

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

// create a course function

const course = {

id: courses.length + 1,

name: req.body.name

// here now we have to enable the json object by default it is not enable so

// to enable this include this command in top

// app.use(express.json()); it's kind of middle ware

};

courses.push(course);

res.send(course);

});

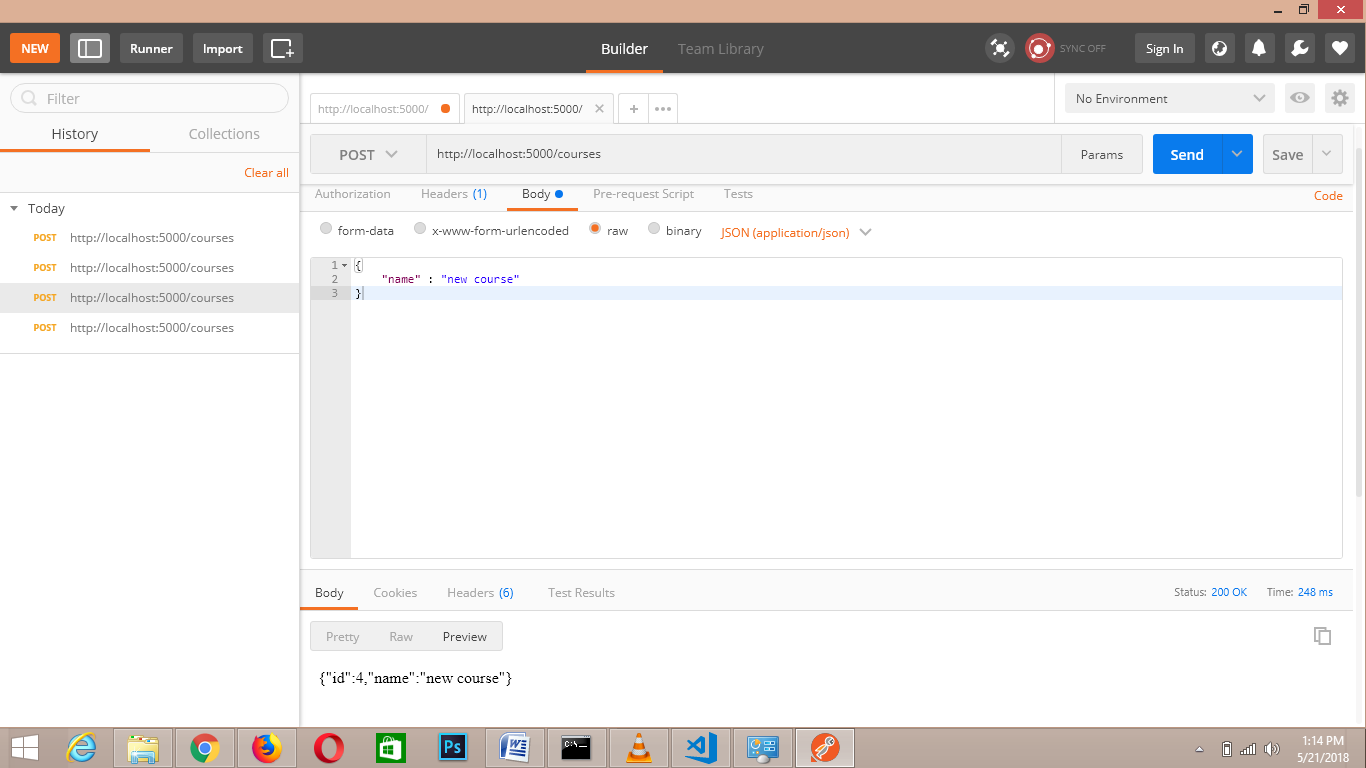
/\*...

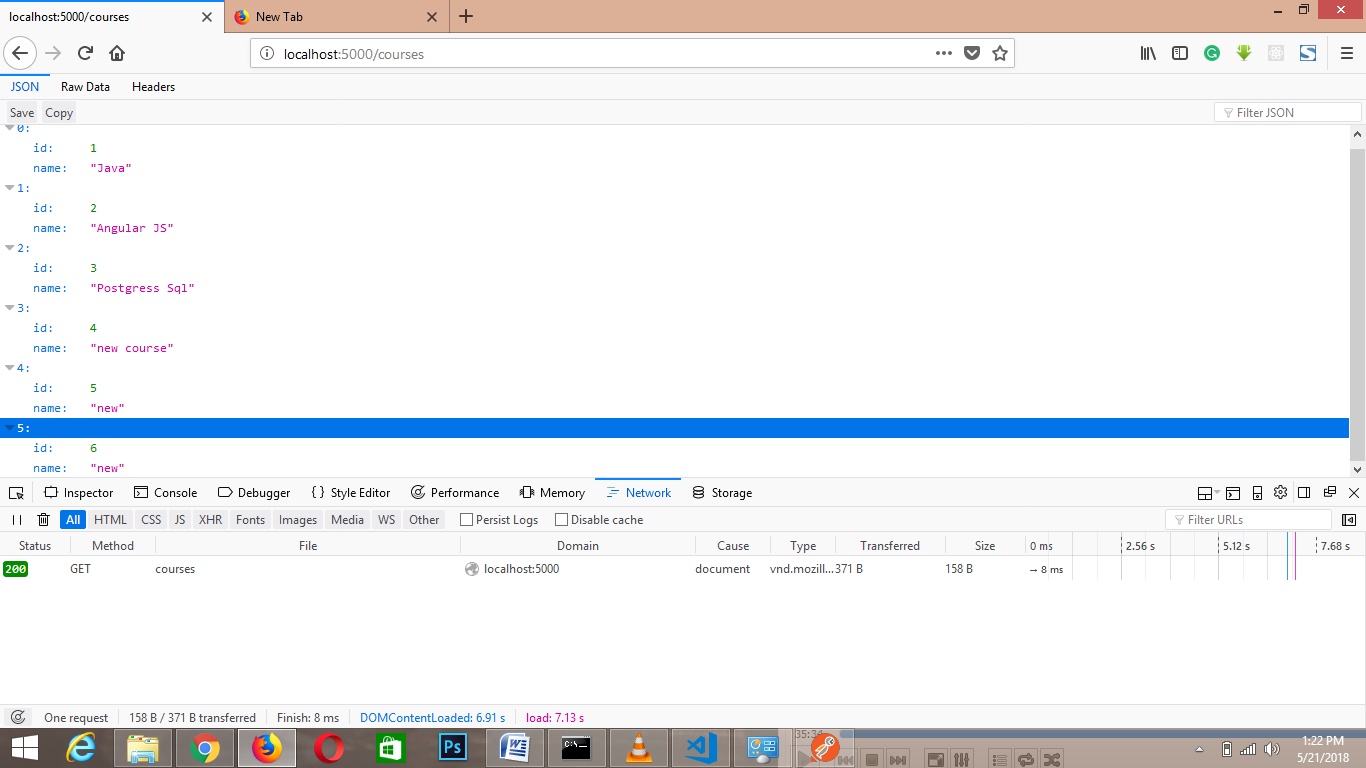
To test the the http post request method

you need to install "postman extension in server"

search for chrome postman

...\*/





Input Validation -

First Method –

You can check the condition - while passing the body request

The syntax is like –

const courses = [

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

{ id: 2, name: 'Angular JS'},

{ id: 3, name: 'Postgress Sql'},

];

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

res.send(courses);

});

// to handle the http post request method

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

//input validation for 404 bad request

if(!req.body.name || req.body.name.length < 3){

// 404 bad r equest

res.status(400).send('Name is required and should be send minimum 3 character');

return;

}

// create a course function

const course = {

id: courses.length + 1,

name: req.body.name

// here now we have to enable the json object by default it is not enable so

// to enable this include this command in top

// app.use(express.json()); it's kind of middle ware

};

courses.push(course);

res.send(course);

});

/\*...

To test the the http post request method

you need to install "postman extension in server"

search for chrome postman

...\*/

Second Method –

Using the “npm module “  
search in google - “npm joi”

Or go to the link – <https://www.npmjs.com/package/joi>

**To install the “npm joi dependencies”**

Run the command in your terminal – “npm intall joi”

And top of the index.js file load the joi module the syntax is like –

Const Joi = require(‘joi’);

// to load the joi module for the input validation

const Joi = require('joi');

The Complete example –

// to load the joi module for the input validation

const Joi = require('joi');

const express = require('express');

const app = express();

app.use(express.json());

// handling the Get Request Method so first define the

// to define the array of computer courses

const courses = [

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

{ id: 2, name: 'Angular JS'},

{ id: 3, name: 'Postgress Sql'},

];

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

res.send(courses);

});

// to handle the http post request method

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

/\*... now define the schema to use the Joi obeject

schema tells the structure of your data

structure means (type, lenght, minimum range, etc)

...\*/

const schema = {

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

};

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

// to check the result in console

console.log(result);

// create a course function

const course = {

id: courses.length + 1,

name: req.body.name

// here now we have to enable the json object by default it is not enable so

// to enable this include this command in top

// app.use(express.json()); it's kind of middle ware

};

courses.push(course);

res.send(course);

});

/\*...

To test the the http post request method

you need to install "postman extension in server"

search for chrome postman

...\*/

To get the same error message on postman –

Edit the code –

// to load the joi module for the input validation

const Joi = require('joi');

const express = require('express');

const app = express();

app.use(express.json());

// handling the Get Request Method so first define the

// to define the array of computer courses

const courses = [

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

{ id: 2, name: 'Angular JS'},

{ id: 3, name: 'Postgress Sql'},

];

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

res.send(courses);

});

// to handle the http post request method

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

/\*... now define the schema to use the Joi obeject

schema tells the structure of your data

structure means (type, lenght, minimum range, etc)

...\*/

const schema = {

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

};

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

//input validation for 404 bad request

if(result.error){

// 404 bad r equest

res.status(400).send(result.error);

return;

}

// create a course function

const course = {

id: courses.length + 1,

name: req.body.name

// here now we have to enable the json object by default it is not enable so

// to enable this include this command in top

// app.use(express.json()); it's kind of middle ware

};

courses.push(course);

res.send(course);

});

/\*...

To test the the http post request method

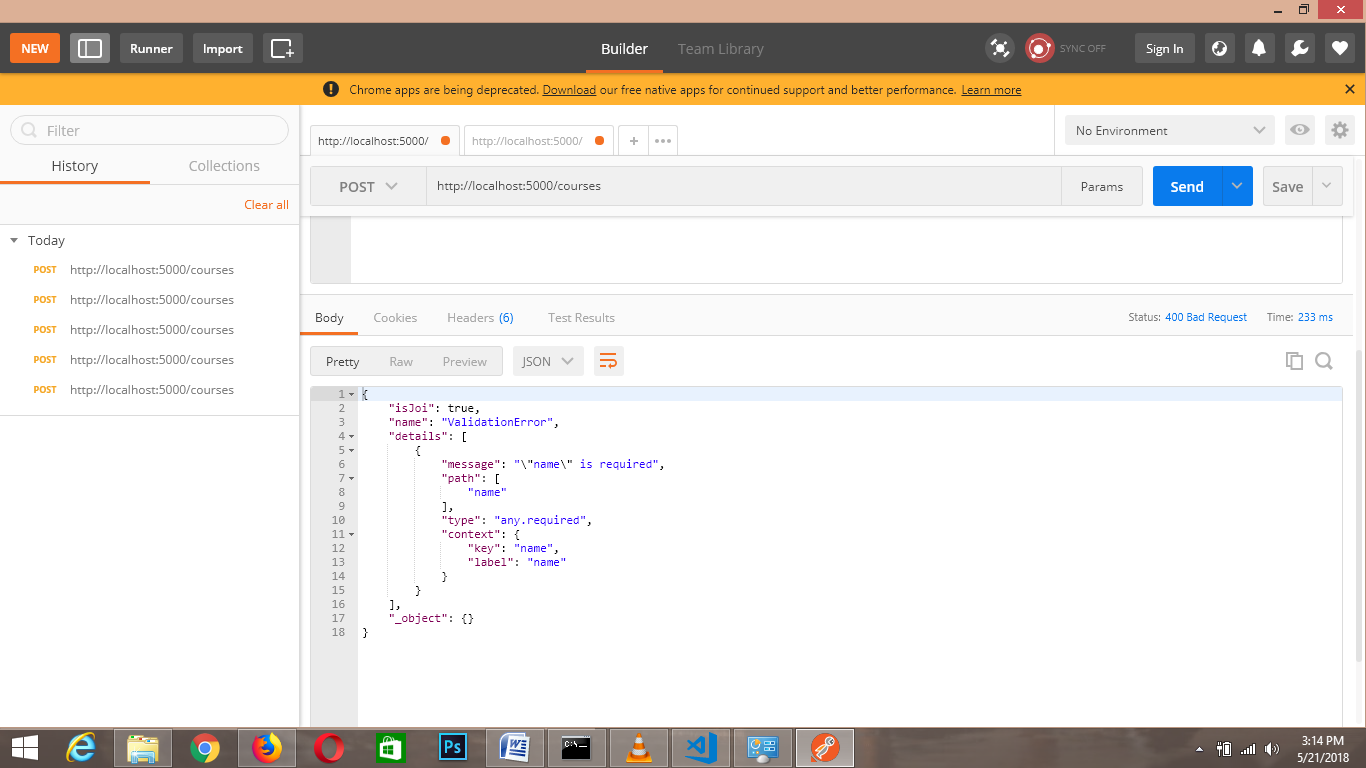
you need to install "postman extension in server"

search for chrome postman

...\*/

Pass the empty value once by the postman

And see the output –



You can even edit this –

if(result.error){

// 404 bad r equest

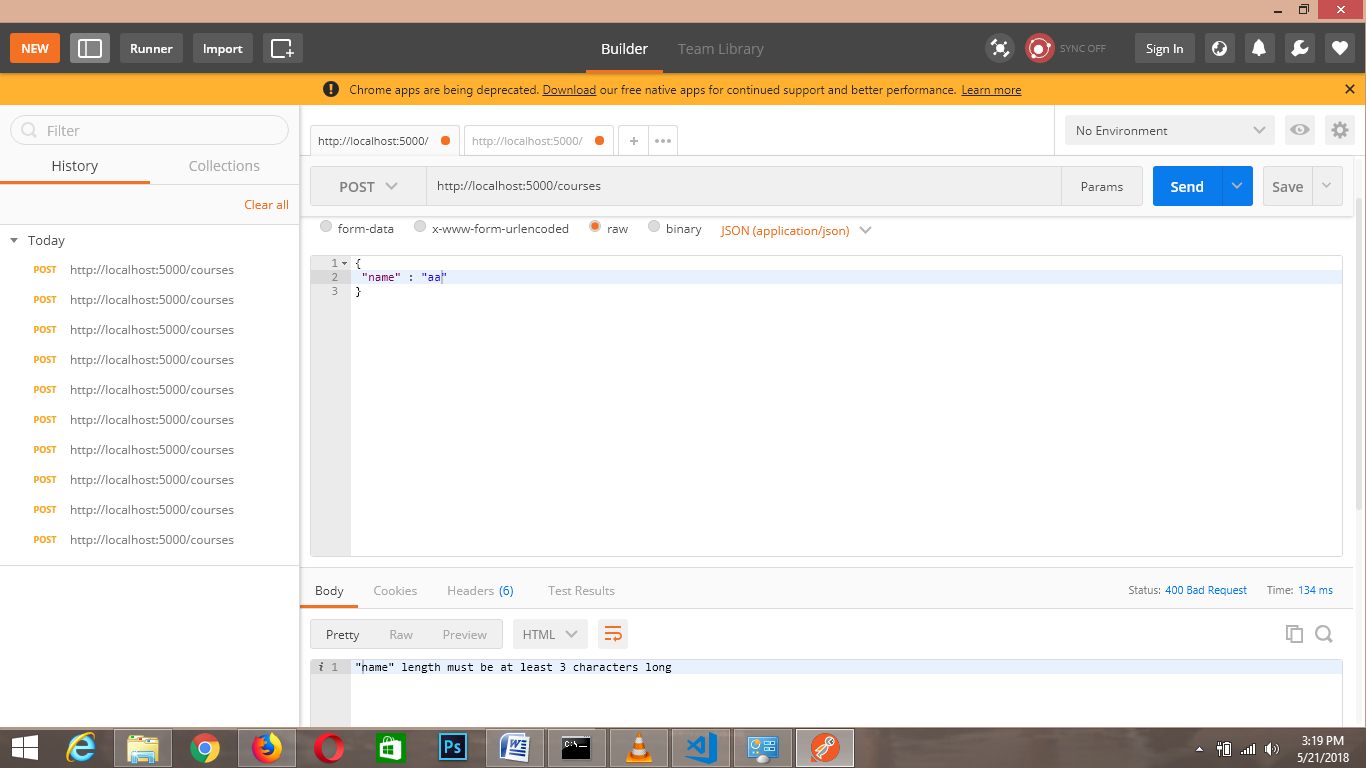
res.status(400).send(result.error.details[0].message);

return;

}

Now you will get the output like this –

“name is required “



Handling the PUT Request -

Basically PUT method is used to update the resources

// to load the joi module for the input validation

const Joi = require('joi');

const express = require('express');

const app = express();

app.use(express.json());

// handling the Get Request Method so first define the

// to define the array of computer courses

const courses = [

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

{ id: 2, name: 'Angular JS'},

{ id: 3, name: 'Postgress Sql'},

];

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

res.send(courses);

});

// to handle the http post request method

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

/\*... now define the schema to use the Joi obeject

schema tells the structure of your data

structure means (type, lenght, minimum range, etc)

...\*/

//input validation for 404 bad request

if(result.error){

// 404 bad r equest

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

return;

}

// create a course function

const course = {

id: courses.length + 1,

name: req.body.name

// here now we have to enable the json object by default it is not enable so

// to enable this include this command in top

// app.use(express.json()); it's kind of middle ware

};

courses.push(course);

res.send(course);

});

/\*...

To test the the http post request method

you need to install "postman extension in server"

search for chrome postman

...\*/

// use of PUT Request to update the course details

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

// implement the logic

// if not existing, return 404

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

if(!course){

res.status(404).send('The course with given ID is not found');

}

// if invalid, return 400 -Bad request

const result = validateCourse(req.body);

if(result.error){

// 404 bad r equest

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

return;

}

// Update course

course.name = req.body.name;

// Return the updated course

res.send(course);

});

// define the validate the function

function validateCourse(course){

// validate

const schema = {

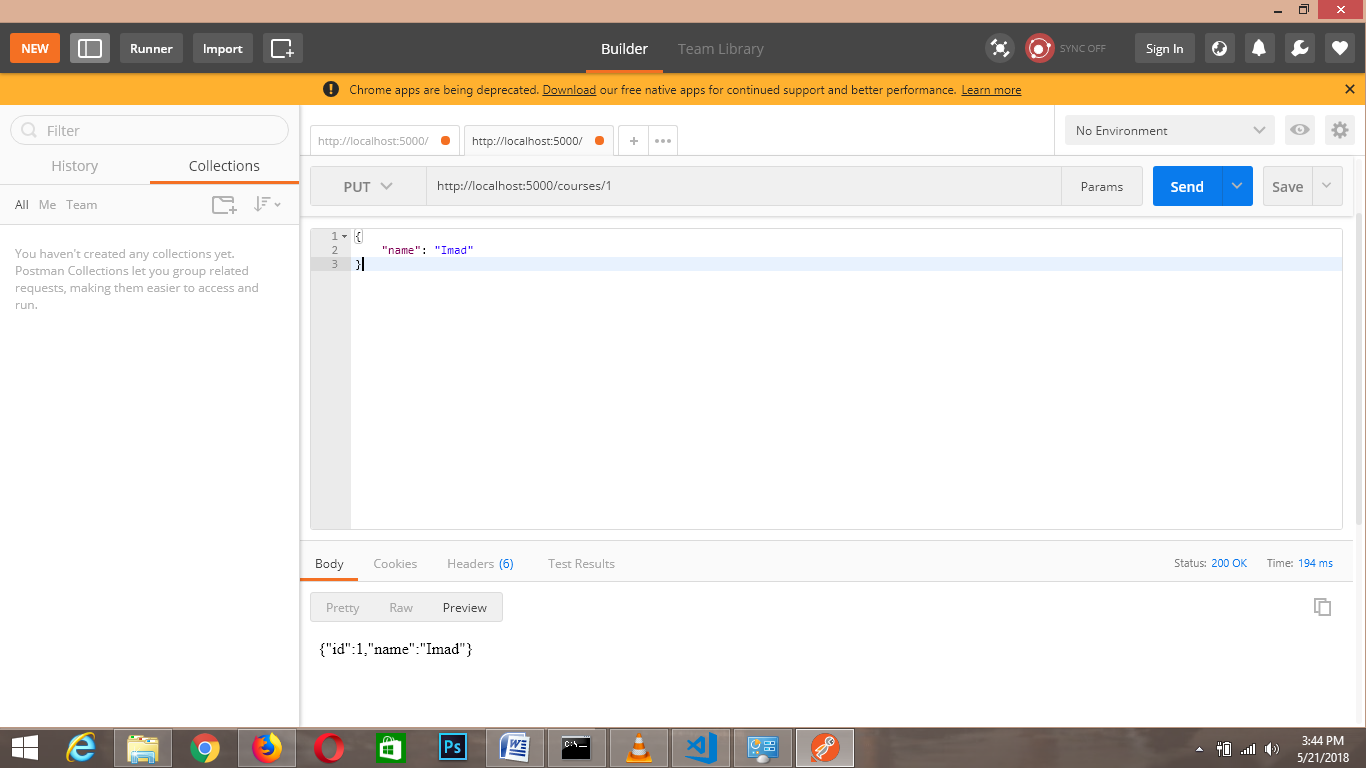
name: Joi.string().min(3).required()

};

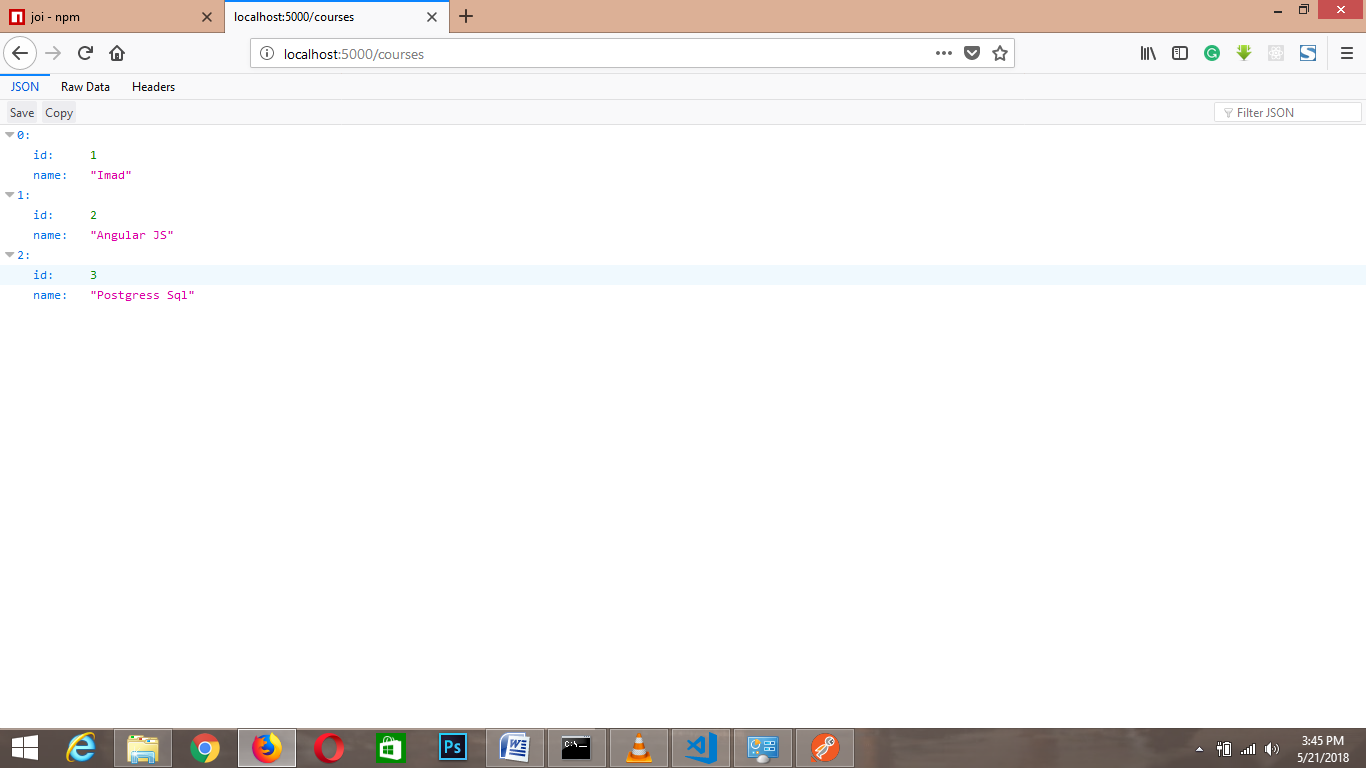
return Joi.validate(course, schema);

}

The Output –



See the updated courses in browser



Handling the DELETE Request -

// handle the DELETE Request

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

// look up the course

// not existing , return 404

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

if(!course){

res.status(404).send('The course with given ID is not found');

}

// Delete

const index = courses.indexOf(course);

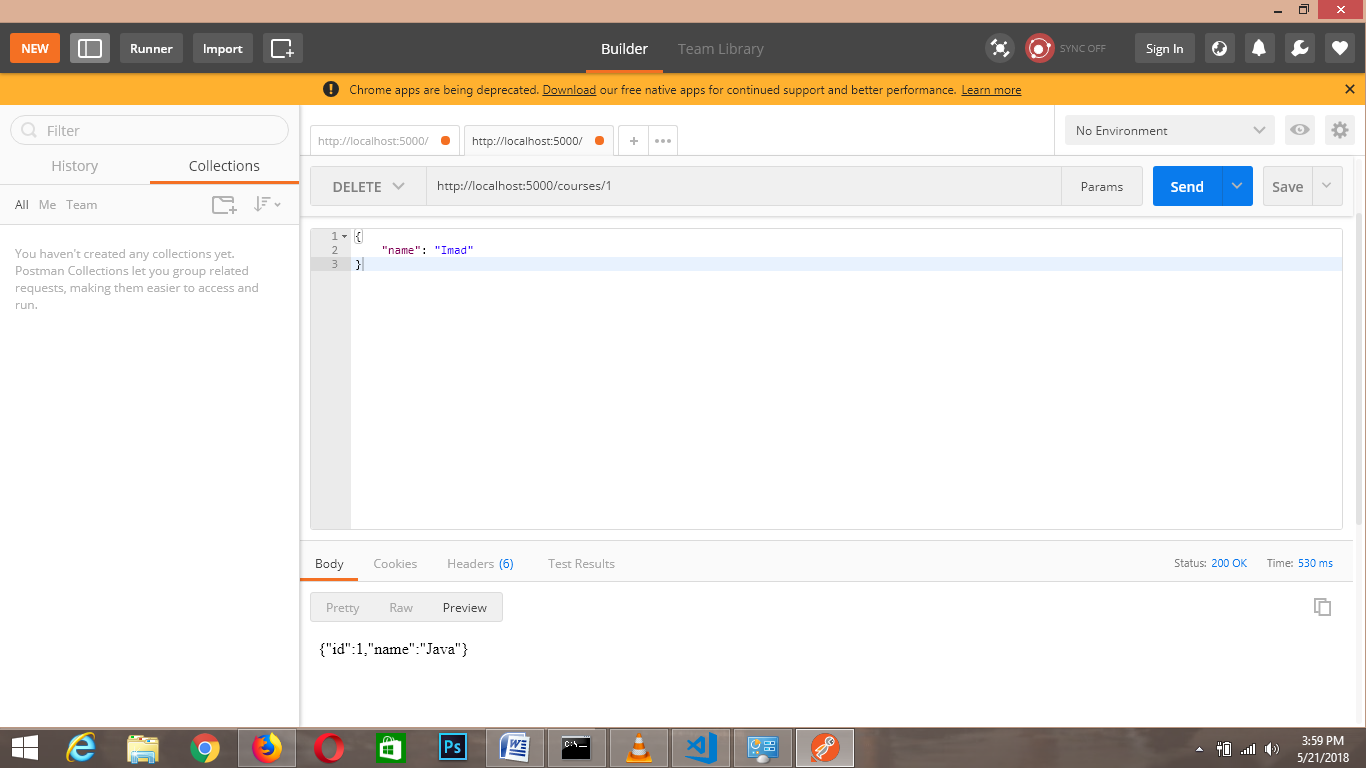
courses.splice(index, 1);

// Return the same course

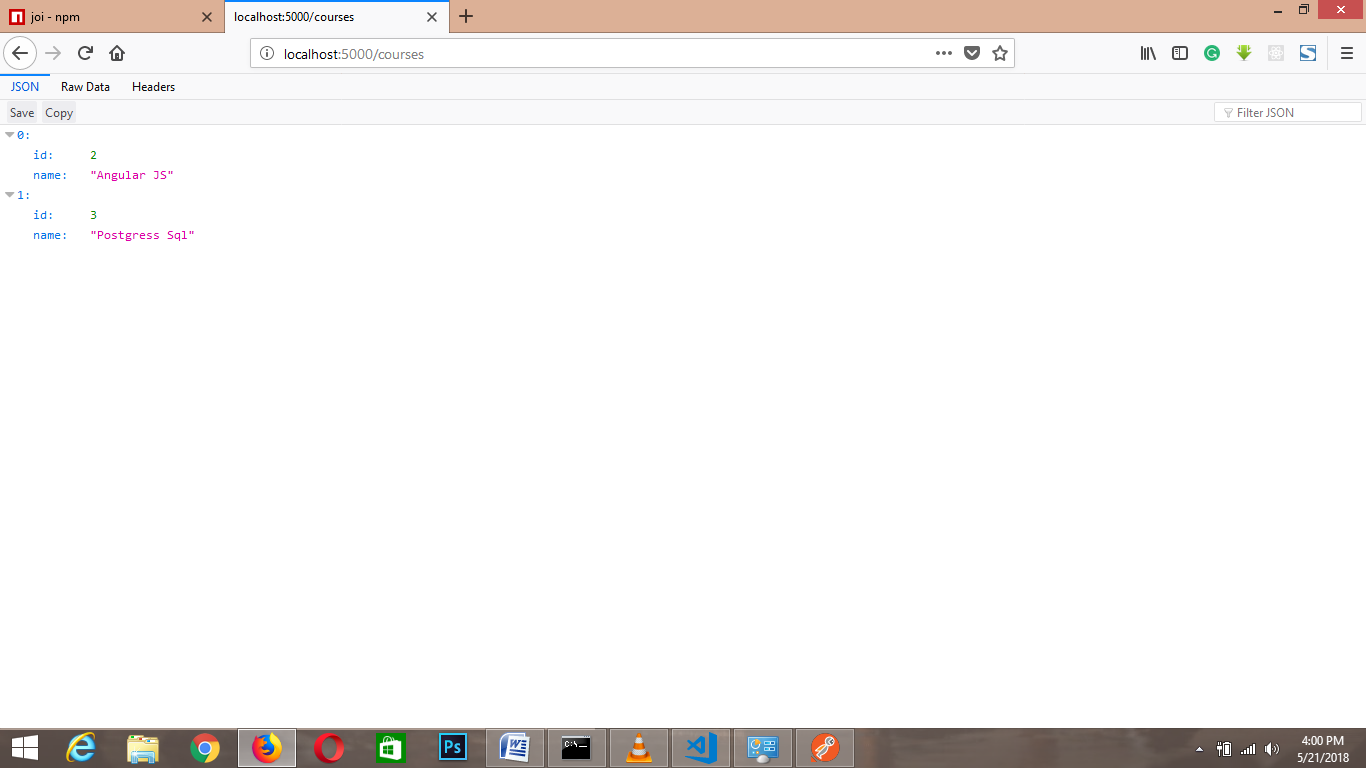
res.send(course);

});

The Output is –



See all the code in browser –



The complete code of App.js file -

// to create the server

// to load the joi module for the input validation

const Joi = require('joi');

const express = require('express');

const app = express();

app.use(express.json());

// handling the Get Request Method so first define the

// to define the array of computer courses

const courses = [

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

{ id: 2, name: 'Angular JS'},

{ id: 3, name: 'Postgress Sql'},

];

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

res.send(courses);

});

// to handle the http post request method

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

/\*... now define the schema to use the Joi obeject

schema tells the structure of your data

structure means (type, lenght, minimum range, etc)

...\*/

//input validation for 404 bad request

if(result.error){

// 404 bad r equest

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

return;

}

// create a course function

const course = {

id: courses.length + 1,

name: req.body.name

// here now we have to enable the json object by default it is not enable so

// to enable this include this command in top

// app.use(express.json()); it's kind of middle ware

};

courses.push(course);

res.send(course);

});

/\*...

To test the the http post request method

you need to install "postman extension in server"

search for chrome postman

...\*/

// use of PUT Request to update the course details

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

// implement the logic

// if not existing, return 404

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

if(!course){

res.status(404).send('The course with given ID is not found');

}

// if invalid, return 400 -Bad request

const result = validateCourse(req.body);

if(result.error){

// 404 bad r equest

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

return;

}

// Update course

course.name = req.body.name;

// Return the updated course

res.send(course);

});

// define the validate the function

function validateCourse(course){

// validate

const schema = {

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

};

return Joi.validate(course, schema);

}

// handle the DELETE Request

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

// look up the course

// not existing , return 404

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

if(!course){

res.status(404).send('The course with given ID is not found');

return;

}

// Delete

const index = courses.indexOf(course);

courses.splice(index, 1);

// Return the same course

res.send(course);

});

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

// emplement some logic

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

if(!course){

res.status(404).send('The course with given ID is not found');

return;

}

else{

res.send(course);

}

})

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

res.send("This is my First Application");

});

// '/' represent the root url

// to fetch the data from url in array of number format

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

res.send([1,2,3,4,5]);

})

/\*... Route Parameters

in above example we set the url '/api/courses' for the no of courses

so what if my endpoint is like this '/api/courses/a' to implement this

idea see the below expample -

...\*/

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

// now request to listen the particular id we need to request the params.id

res.send(req.params.id);

});

// Anohter example of route parameter

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

res.send([

"Monika",

"harika",

"Sankar",

"Aakash",

"Khagesh"

])

});

// to create route parameters for the friends

app.get('/friends/:name', (req,res) =>{

res.send(req.params.name);

})

// set the multiple parameter in your route to bulid a complex application

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

res.send(req.query);

})

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

res.send("Hello world");

})

// to assign the port no dynamically because when you will host the application your port no

// will assing dynamically so you need to change here

// express js gives the environment variable called port to assign the port no dynamically.

// PORT

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

/\*....

NOTE- here process is the global object and it has the propety called env(environment)

and after that we are adding here name of he environment variable which is PORT

if it is set then port no will assing automatically if it is not

then port no will assign 3000.

Now make some changes in app.listen method(port, () => console.log("Listening port on ${port}"));

instead of "" in console.log("") use single back tag (`........`);

....\*/

// to listen the port on the server

app.listen(port, () => console.log(`listening port on ${port}`));