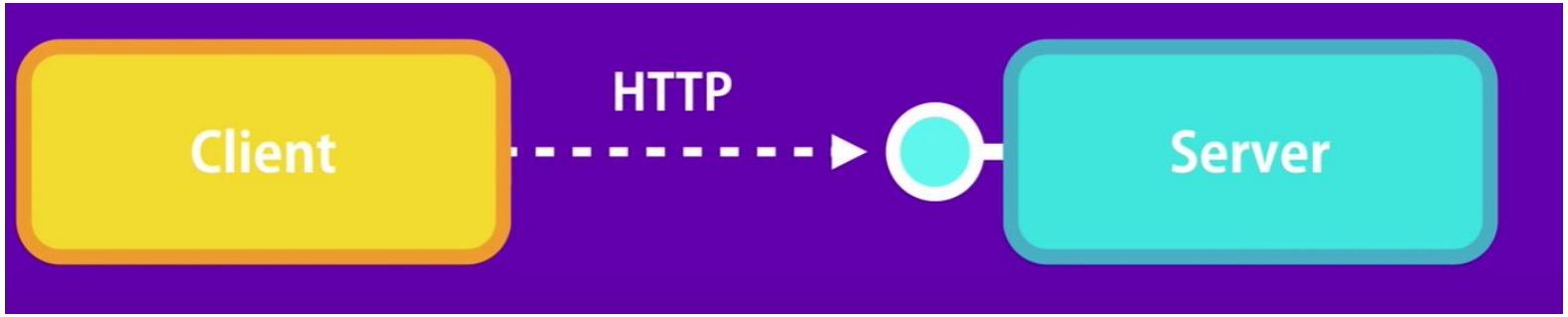
REST API with Node JS and Express

What is RESTful Service

Client-Server Architecture: Client(front-end part) and Server(back-end part). Server offers various services. The client call these services by HTTP request.



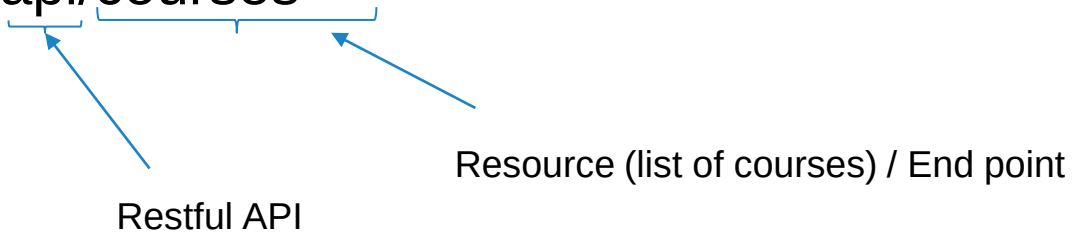
REST: Representational State Transfer: A convention to provide HTTP services for CRUD operations.

- Create
- Read
- Update
- Delete

Image Source: https://www.youtube.com/watch?v=pKd0Rpw7O48

What is RESTful Service

http://example.com/api/courses



All the CRUD operations will be done by sending a HTTP request to the end point. HTTP methods specify the kind of the operation.

HTTP Methods:

GET: get data
POST: create data
PUT: update data
DELETE: delete data

HTTP Methods

HTTP Methods		Request	Response
GET	To get all the course	GET /api/courses	[{ "id": 1, "name": "Maths" }, { "id": 2, "name": “Science" },] Returns array of customer objects
	To get a course	GET /api/courses/1	{ "id": 1, "name": "Maths" },
POST		POST /api/courses {name: ‘’}	{ "id": 3, "name": ‘’ }
PUT	To update a course	PUT /api/courses/1 {name: ‘’} // Include course object that needs to update in the body of request	{ "id": 1, "name": ‘’ }
DELETE	To delete a course	DELETE /api/courses/1	{ "id": 1, "name": ‘’ }

The resources are available using meaningful address

Introducing Express

is a back end web application framework for Node.js.

designed for building web applications and APIs.

It has been called the de facto standard server framework for Node.js.

To Install:

$cd express_demo

express_demo $npm init --yes

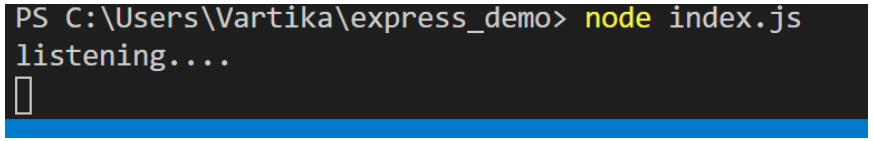
express_demo $npm i express

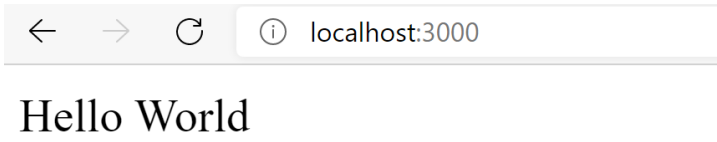
Web Server

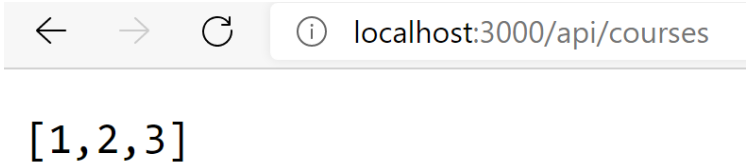
```
const express=require('express');
const app=express(); // provides get, post, update and delete methods

app.get('/', (req,res) => {// takes root path and callback function which takes two arguments request an
d response
res.send('Hello World');

});
app.get('/api/courses', (req,res)=>{
res.send([1,2,3]);  // array can be replaced with actual course object
});
Const port=process.env.PORT || 3000; // environment variable
app.listen(port, ()=> console.log('listening....'));
//app.listen(3000, ()=> console.log('listening....'));
```







express_demo $npm i –g nodemon

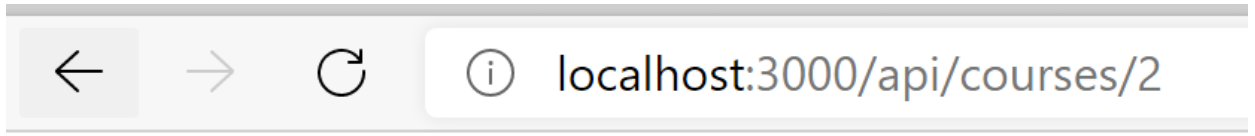
Express_demo$nodemon index.js // Changes will reflect automatically, no need to stop the server

Route Parameters

How to create a route to get single value/course?
/api/course/1 where 1 is the id of the course.

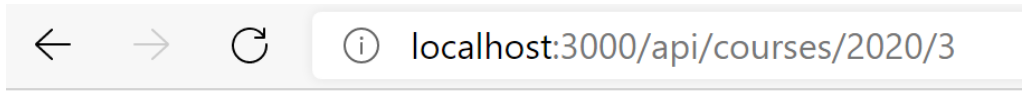
Any literal

```
app.get('/api/courses/:id', (req,res)=>{
res.send(req.params.id);
});
```



2

```
app.get('/api/courses/:year/:month', (req,
res.send(req.params);
});
```



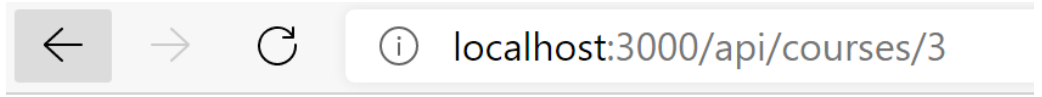
{"year":"2020","month":"3"}

Handling Get Requests

```
//// Array of objects
const courses=[
    {id:1, name:'Maths'},
    {id:2, name:'English'},
    {id:3, name:'Physics'},
    {id:4, name:'Computer'},
]
app.get('/api/courses', (req,res)=>{
    res.send(courses);  // array can be replaced with actual course object
    });
```

[{"id":1,"name":"Maths"},{"id":2,"name":"English"},{"id":3,"name":"Physics"},{"id":4,"name":"Computer"}]

```
app.get('/api/courses/:id', (req,res)=>{
        const course=courses.find(c => c.id===parseInt(req.params.id));
        if(!course) rerurn res.status(404).send('Course not found');
        res.send(course);
        });
```



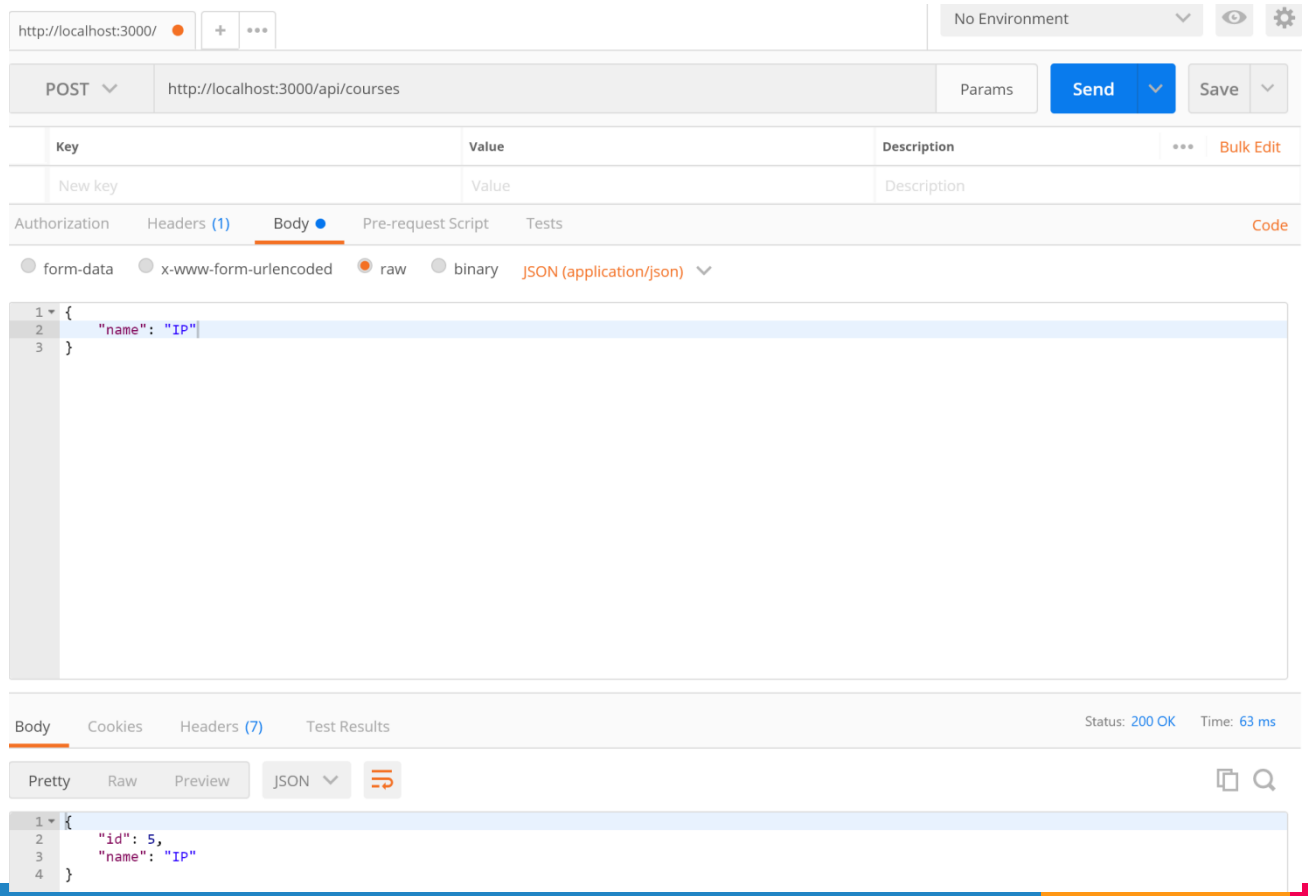
{"id":3,"name":"Physics"}

Handling POST Requests

```
app.use(express.json()); // to parse json


app.post('/api/courses', (req,res)=>{
const course={id: courses.length+1,
name: req.body.name
};
courses.push(course);
res.send(course);
});
```

Use POSTMAN chrome extension.

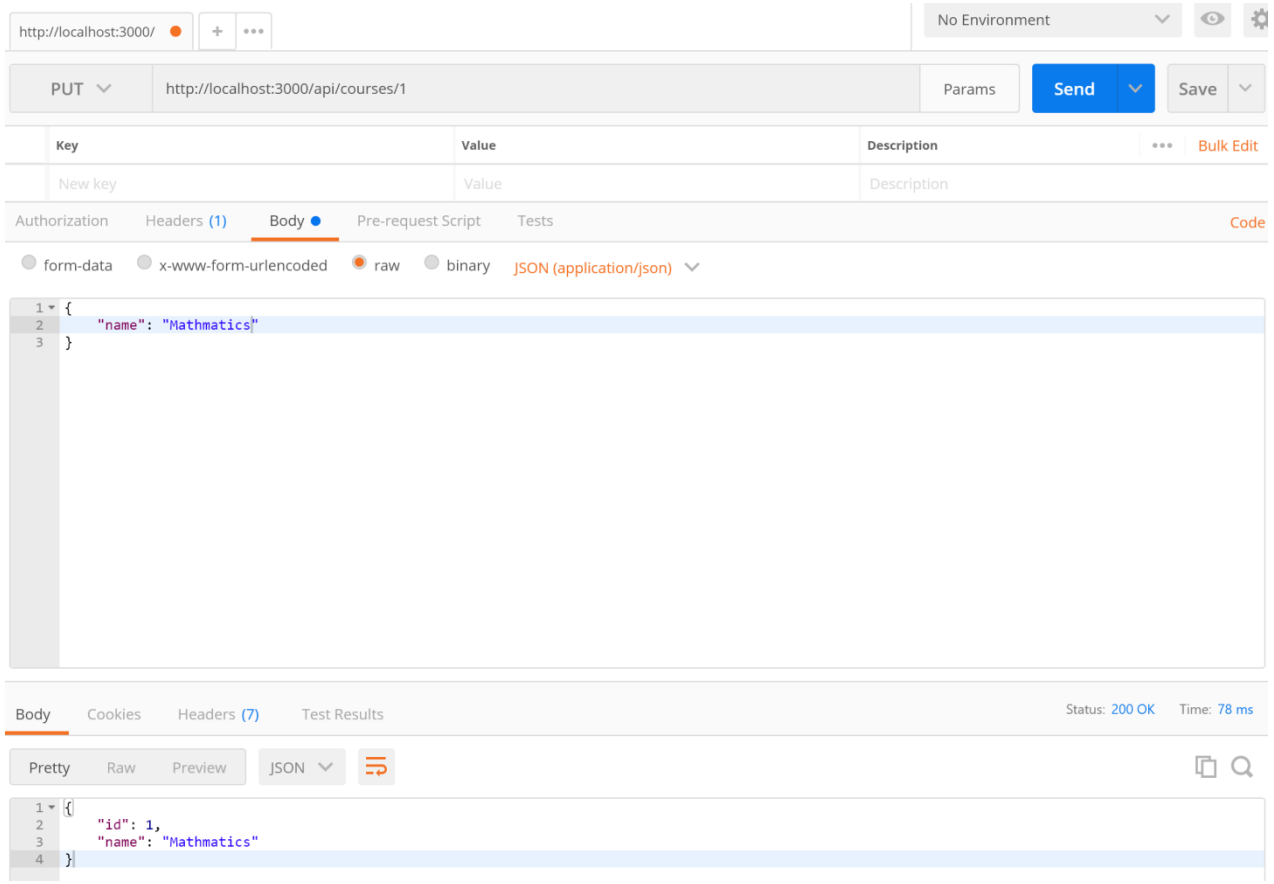


Handling PUT Requests

1. Look up the course
2. if not existing, return 404
3. Update course
4. Return the updated course

```
app.put('/api/courses/:id', (req, res)=>{
    //Look up the course
    // if not existing, return 404
    const course=courses.find(c => c.id===par
seInt(req.params.id));
    if(!course) rerurn
res.status(404).send('Course not found');
    // Update course
    course.name=req.body.name;

    // Return the updated course
    res.send(course);
});
```



Handling Delete Requests

```
app.delete('/api/courses/:id', (req, res)=>{
    const course=courses.find(c => c.id===parseInt(req.params.id));
    if(!course) return res.status(404).send('Course not found');

    // delete
    const index=courses.indexOf(course);
    courses.splice(index, 1);

    //return
    res.send(course);
});
```

// array.splice(index, howmany)

Six Key Constraints

1. Client-Server

- the client and the server should be separate from each other and allowed to evolve individually and independently.
- For example, I should be able to make changes to my mobile application without impacting either the data structure or the database design on the server. Similarly, I should be able to modify the database or make changes to my server application without impacting the mobile client.

2. Stateless

- REST APIs are stateless, i.e. calls can be made independently of one another, and each call contains all of the data necessary to complete itself successfully.
- For example, API key, access token, user ID, etc.
- In order to reduce memory requirements and keep your application as scalable as possible, a RESTful API requires that any state is stored on the client—not on the server(not store any session data).
- Improves scalability.
- Requires more bandwidth.

3. Cache

- Response should be cacheable if possible.
- It requires that every response should include whether a response can be cacheable or not.

Six Key Constraints

4. Uniform Interface

- To decouple client from server, there is need to have a uniform interface that allows independent evolution of the application without having the application's services, models, or actions tightly coupled to the API.
- Single language and standardized means of communicating between the client and the server, such as using HTTP with URI resources, CRUD (Create, Read, Update, Delete), and JSON.

5. Layered System

- Layered system is a system comprised of layers, with each layer having a specific functionality and responsibility.
- REST API design, follows the principle same as Model View Controller, with different layers of the architecture working together to build a hierarchy that helps create a more scalable and modular application.
- Freedom to move systems in and out of your architecture as technologies and services evolve,
- Increasing flexibility and longevity.
- Enhances security since it allows you to stop attacks at the proxy layer, or within other layers, preventing them from getting to your actual server architecture.

6. Code on Demand

- Optional Constraint.
- In addition to data, the servers can provide executable code to the client, eg. Javascript etc.
- Reduces security and visibility.

Useful Reference Links

- https://nodejs.dev/learn