**Node.js**

Node.js is a javascript running on the server.

**History of Node.js**

In 2009 an idea came in the mind of Ryan Dahl (Google Engineer) he tool V8 engine and embedded in a c++ program and called it Node.exe later on became Node.js.

V8 engine + c++ code = Node.exe

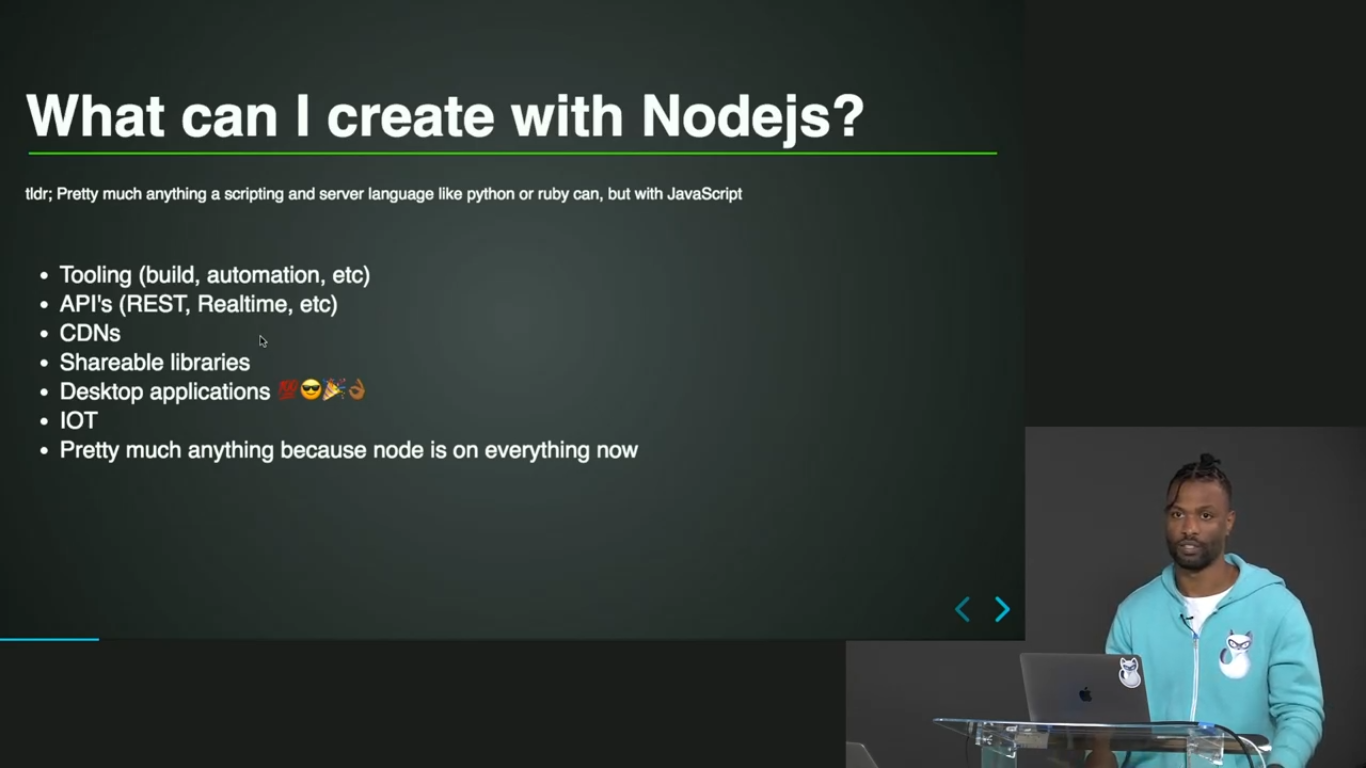
**Note** : V8 is a javascript interpreter developed by google written in c++

**Features of Node.js**

* Node.js uses JavaScript on the server
* Node.js is an open source
* Node.js is free
* Node.js is asynchronous (i.e. does not wait)
* Node.js is a single threaded language which in background uses multiple threads
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)

**When to use Node.js**

1. Input/ Output bound Applications
2. Data Streaming Applications
3. JSON APIs based Applications



**How Node.js handles a file request:**

* Sends the task to the computer's file system.
* Ready to handle the next request.
* When the file system has opened and read the file, the server returns the content to the client.

**Note :** Node.js eliminates the waiting, and simply continues with the next request.

**REPL**

* REPL stands for Read-Eval-Print-Loop.
* It can be used to run/debug the javascript code.

Read − Reads user's input, parses the input into JavaScript data-structure, and stores in memory.

Eval − Takes and evaluates the data structure.

Print − Prints the result.

Loop − Loops the above command until the user presses ctrl-c twice.

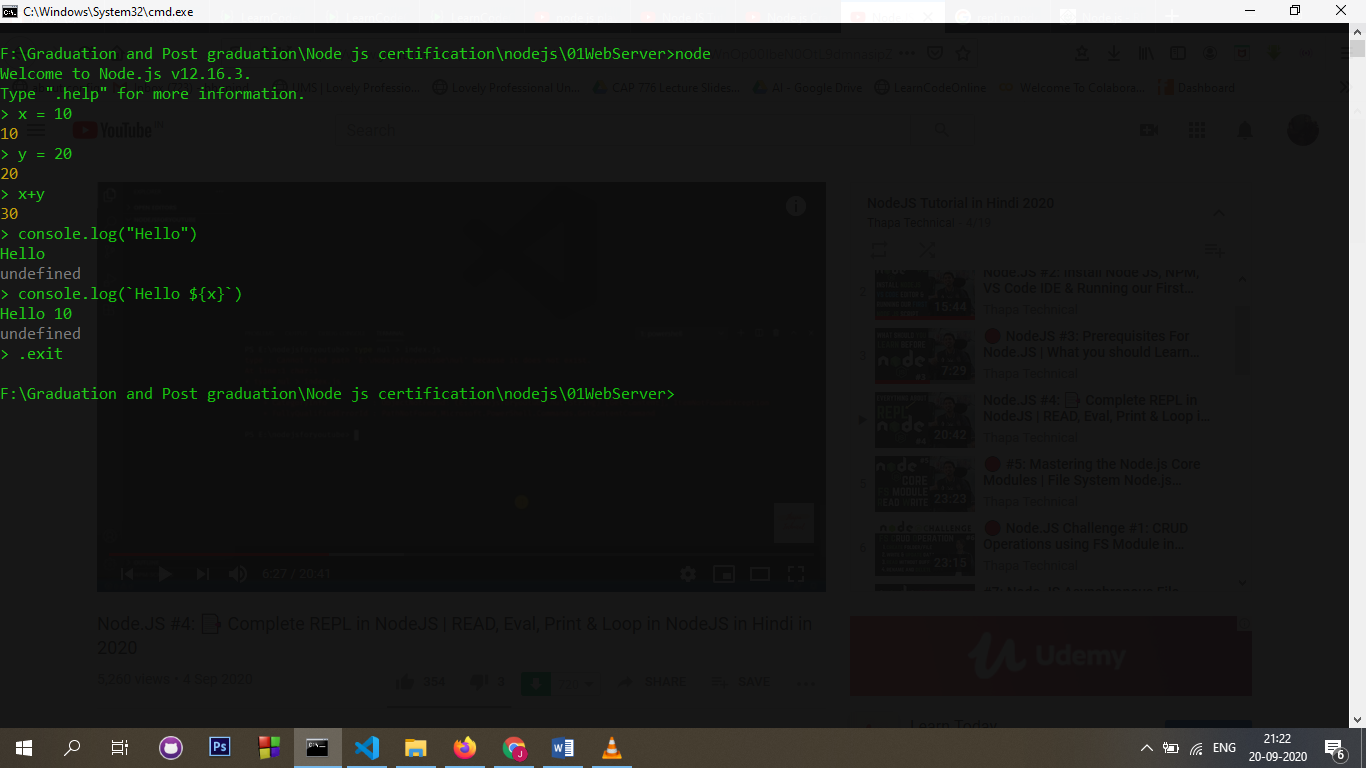
**Steps to start REPL**

step1.) Open command prompt

step2.) type node (now we are inside REPL)

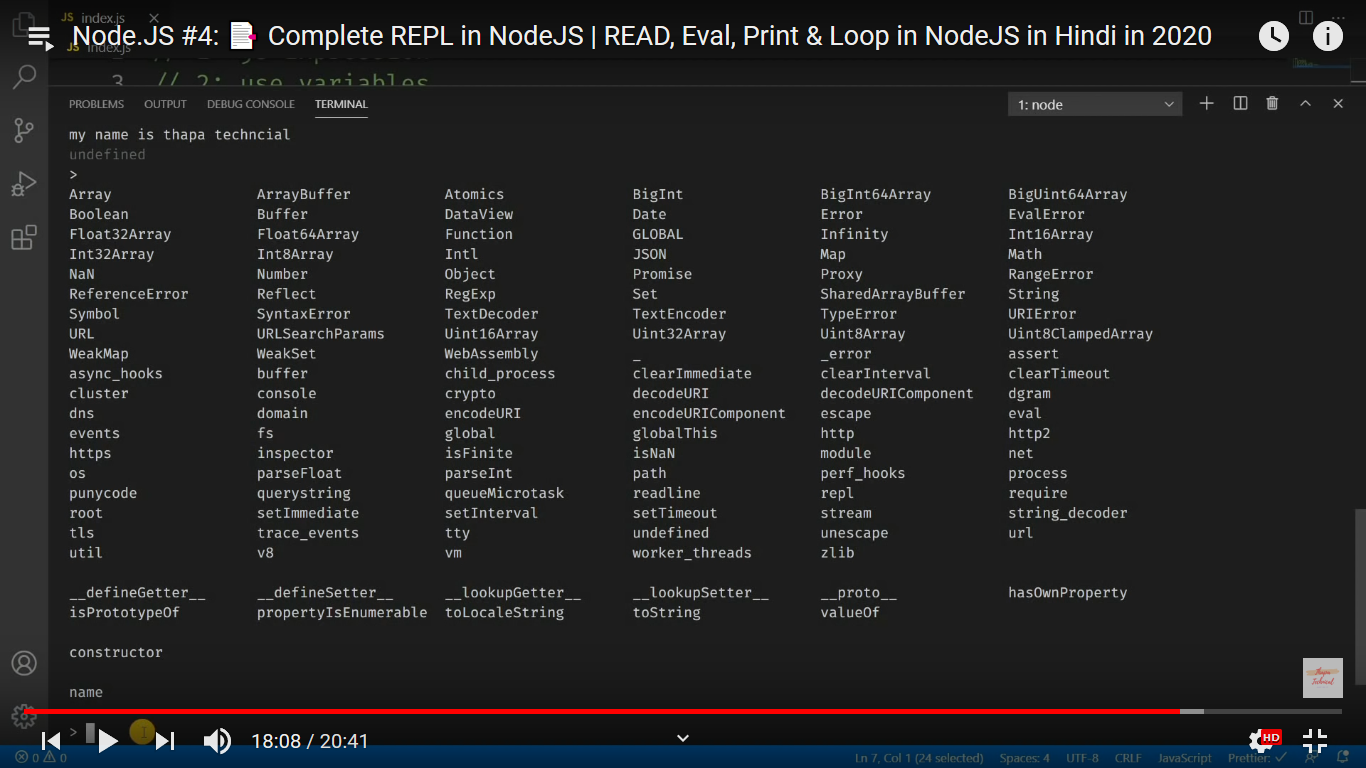
step3.) Now you can run the commands

**Examples**



**How to check the properties and methods available in node.js**

Type tab (twice) inside the REPL



REPL Commands

* **ctrl + c** − terminate the current command.
* **ctrl + c twice** − terminate the Node REPL.
* **\_ (underscore)** - use to get the last result.
* **.help** − list of all commands.
* **.editor** – to enter in the editor mode
* **.break** − exit from multiline expression.
* **.save *filename*** − save the current Node REPL session to a file.
* **.load *filename*** − load file content in current Node REPL session.
* **tab Keys** − list of current commands.

**Module**

Every file in node application is consider as module. The variables and functions were defined in that file (or that module) are scoped to that file.

Means they are private

By default they are private because node.js wrap those variable and functions inside a function known as Module wrapper function.

**Syntax (Module wrapper function)**

( function(exports, require, module, \_\_filename, \_\_dirname) // Anonymous function

{

// node.js puts our code in here

} );

Note : By doing this, Node.js achieves a few things:

* It keeps top-level variables (defined with var, const or let) scoped to the module rather than the global object.
* It helps to provide some global-looking variables that are actually specific to the module, such as:
* The module and exports objects that the implementor can use to export values from the module.
* The convenience variables \_\_filename and \_\_dirname, containing the module's absolute filename and directory path.

**Note : If we want to use a variable or function defined in a file (or module) outside that module then we need to explicitly export it and make it public**

**There are three types of modules**

**1.) core modules (built in modules) : Don't need to install**

**2.) Local modules (the .js file we created in our node application is called Local module)**

**3.) Third party modules (modules that are created by others) : we can use install third party modules by using npm**

**Node.js Modules**

Modules simply mean built-in libraries.

**How to create own module**

function add(n1, n2)

{

    console.log("Addition = "+(n1+n2));

}

function sub(n1, n2)

{

    console.log("Subtraction = "+(n1-n2));

}

// Make this function public so that other files can use this (i.e Export the module)

// module.exports.add = add;

// module.exports.sub = sub;

// OR

module.exports = {add, sub};

// Import the module

const ao = require("./arithmeticoperation");

console.log(ao);            // ao now acting as object & add, sub are its method

// Use it

ao.add(10,5);

ao.sub(10,5);

**Some Important Core Modules are**

1.) File system

2.) HTTP

3.) URL

**File system**

The Node.js file system module allows you to work with the file system on your computer.

// modules can be loaded by the Node.js require() method of module object.

const fs = require(“fs”);

**How to create a file and write into the file**

writeFileSync() method is used to create a file and write into the file.

**Note** : If the file is already present it will overwrite the content

If we don’t want to overwrite the file appendFile() method of File system module

**Example**

const fs = module.require(“fs”);

fs.writeFileSync(“textfile.txt”, “Programming is my passion”);

**How to read files**

readFile() method is used to read files on your computer.

**Example**

const data = fs.readFileSync("textfile.txt",{encoding:'utf8', flag:'r'});

console.log(data);

const data = fs.readFileSync("hello.txt");

console.log(data.toString());

**How to rename the file**

renameSync() : renameSync() method is used to rename the file.

**Example**

fs.renameSync("textfile.txt","newtextfile.txt");

**1.) File system**

// (fs) File System Module

const fs = require("fs");

// Create folder

// fs.mkdirSync("Practice nodejs");

// Create a file and write in the file

// fs.writeFileSync("hello.txt","hello frnds i am learning nodejs");

// Note : If the file with this name is not present then it will create a new file and write the data in the file but If the file is already present then it will overwrite the data

// append the new data in the file

// fs.appendFileSync("hello.txt","I love you bro keep doing hardwork");

// Read the file

// const data = fs.readFileSync("hello.txt","utf8");

// console.log(data);

// Note : In readFileSync() we have to specify the encoding type because if we don't specify the encoding type then it will return buffer data

// OR

// const data = fs.readFileSync("hello.txt");

// console.log(data.toString());

// Rename the file

// fs.renameSync("hello.txt","learningfilesystem.txt");

// Delete a file

// fs.unlinkSync("learningfilesystem.txt");

// Delete a folder

// fs.rmdirSync("Practice nodejs");

// Asynchronous

// asynchronous means parallelly (i.e. it will not wait it will keep taking the request)

// For understanding asynchronous we have to first understand call back function

// Callback function : when a function passed into another function as an argument

// which is then called inside the body of another function

// Note : This technique allows a function to call another function

// Example :

// funA() is the callback function

// function funA()

// {

//     console.log("i am called from inside a funB");

// }

// function funB(callback)

// {

//     console.log('do something');

//     if(typeof(callback==='function'))

//     {

//         //callback function (i.e. funA) is called

//         callback();

//     }

//     else

//     {

//         console.log('Argument you are passing must be function');

//     }

// }

// we are passing the function as argument (this is called callback function)

// funB(funA);

// first funB is called AND funB will finished its execution

// then funA is called

// funA is saying to funB please funB call me back when you complete your execution

// Create directory

// fs.mkdir("nodejs",(err) => {

//     console.log("Folder created");

//     console.log(err);

// });

// Create a file and write in the file

// fs.writeFile("bye.txt","I love programming. It is my passion.", (err) => {

//     console.log("Task completed");

//     console.log(err);                        // tells about error

// } );

// Read the file

// fs.readFile("learningfilesystem.txt","utf8",(err, data) => {

//     console.log(data);

//     console.log("Task completed");

//     console.log(err);

// });

2**.) URL**

// URL Module

// The URL module splits up a web address into readable parts.

// import

const url = module.require('url');

// requested url

const requrl = 'http://www.gmail.com/login?country=India&state=Punjab';

// Parse an address with the url.parse() method, and it will return a URL object

const urlObject = url.parse(requrl,true);

console.log(urlObject);

// host : It is used to get get and set the host portion of the URL.

console.log(urlObject.host);

// pathname : It is used to get and set the path portion of the URL.

console.log(urlObject.pathname);

// search : It is used to get and set the serialized query portion of the URL.

console.log(urlObject.search);

**3.) Operating system**

// (OS) Operating System module

// The OS module provides information about the computer's operating system.

const os = require("os");

// Methods

// arch() : Returns the operating system CPU architecture

console.log(os.arch());

// hostname() : Returns the hostname of the operating system

console.log(os.hostname());

// type() : Returns the name of the operating system

console.log(os.type());

// totalmem() : Returns the number of total memory (RAM) of the system

console.log("Total RAM = "+os.totalmem());   // output will be in bytes

// freemem() : Returns the number of free memory (RAM) of the system

console.log("Free RAM = "+os.freemem());     // output will be in bytes

// In GB

let fm = os.freemem();

fm = fm/1024/1024/1024;

console.log("Free RAM (in gb) = "+fm);

**4.) Path**

// Path Module

// The Path module provides a way of working with directories and file paths.

const path = require("path");

// Methods

// dirname(filename) :  It return the name of the directory

console.log(path.dirname("F:/Graduation and Post graduation/Node js certification/Practice nodejs/practicepath.js"));

// basename(filename) : It returns name of the file with extension

console.log(path.basename("F:/Graduation and Post graduation/Node js certification/Practice nodejs/practicepath.js"));

// extname(filename) : It Returns only the extension of the file

console.log(path.extname("F:/Graduation and Post graduation/Node js certification/Practice nodejs/practicepath.js"));

// parse(filename) : The path.parse() method is used to return an object whose properties represent the given path. This method returns the following properties:

// root (root name)

// dir (directory name)

// base (filename with extension)

// ext (only extension)

// name (only filename)

// Note : It returns a object

// Note : Inside object we can have properties or function

console.log(path.parse("F:/Graduation and Post graduation/Node js certification/Practice nodejs/practicepath.js"));

// How to access properties

const myPath = path.parse("F:/Graduation and Post graduation/Node js certification/Practice nodejs/practicepath.js");

console.log(myPath);

console.log(myPath.root);

console.log(myPath.dir);

console.log(myPath.name);

console.log(myPath.base);

console.log(myPath.ext);

**Web Server**

// Web server is a software that serves web content through the http protocols

// Example : IIS is a web server for ASP.NET web applications and Apache is a web server for PHP or Java web applications

// With Node.js we can create our own web server which will handle HTTP requests asynchronously.

// Node.js has a built-in module called HTTP which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

// HTTP module : HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

// Create web server with http module of node.js

// Import the module

const http = require("http");

// call the createServer() method : This method is used to create server

// This method takes one callback function as argument

// This callback function takes two parameters (1.) request object (2.) response object

// HTTP request is handled by request object

// HTTP response is handled by response object

const server = http.createServer(function (req, res)

{

    // Request

    console.log(`Requested page = ${req.url}`);         // We can use url property of request object to get the url from where the request was made

    // Message (response) sent to the request made by the client

    res.end("Hello from the other side bhupinder paji");

});

// listen() : The listen() method is used to listen the request

// syntax : server.listen(port, hostname, backlog, callback);

server.listen(8000, "127.0.0.1", function ()

{

    console.log("Listening to the port no 8000");

});

**Routing**

****

**Example**

// Routing : Routing defines the way in which the client requests are handled.

// Create web server with http module of node.js

// Import the module

const http = require("http");

// call the createServer() method : This method is used to create server

// This method takes one callback function as argument

// This callback function takes two parameters (1.) request object (2.) response object

const server = http.createServer(function (req, res)

{

    //request

    console.log(`Requested page = ${req.url}`);         // We can use url property of request object to get the url from where the request was made

    // Routing

    if(req.url == "/" || req.url== "/home")

    {

        // Message (response) sent to the request made by the client

        res.write("Hello from the home sides");

        // The end() function is used to end the response process.

        res.end();

    }

    else if(req.url == "/signup" || req.url == "/login")

    {

        //Message (response) sent to the request made by the client

        res.write("Hello from the signup/login sides");

        // The end() function is used to end the response process.

        res.end();

    }

    else if(req.url == "/about")

    {

        //Message (response) sent to the request made by the client

        res.write("Hello from the aboutus sides");

        // The end() function is used to end the response process.

        res.end();

    }

    else

    {

        //Message (response) sent to the request made by the client

        res.writeHead(404, {"Content-type" : "text/html"});

        res.write("<h1> 404 error. Page doesn't exist </h1>");

        // The end() function is used to end the response process.

        res.end();

    }

});

// listen() : The listen() method is used to listen the request

// syntax : server.listen(port, hostname, backlog, callback);

server.listen(8000, "127.0.0.1", function ()

{

    console.log("Listening to the port no 8000");

});

// HTTP response status codes

// ------------

// Informational responses (100–199)

// Successful responses (200–299)

// Redirects (300–399)

// Client errors (400–499)

// Server errors (500–599)

**API**

-> API stands for application programming interface

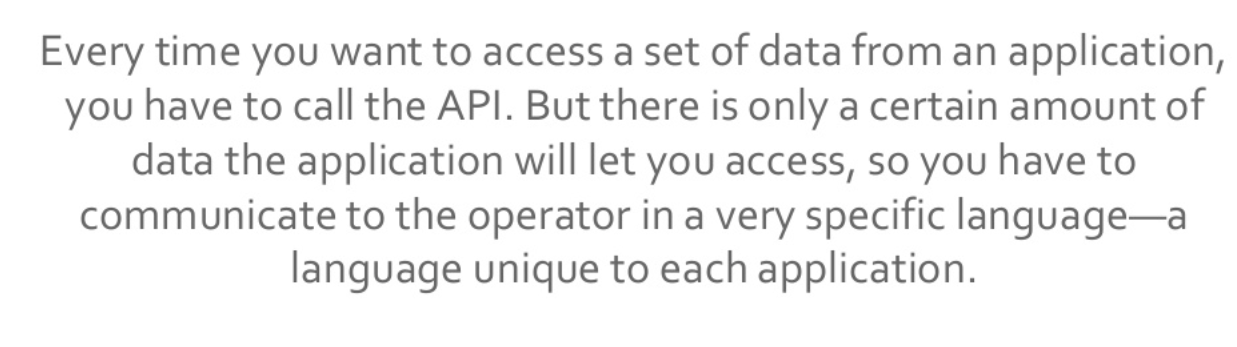
-> Actually it is the layer between the client and database

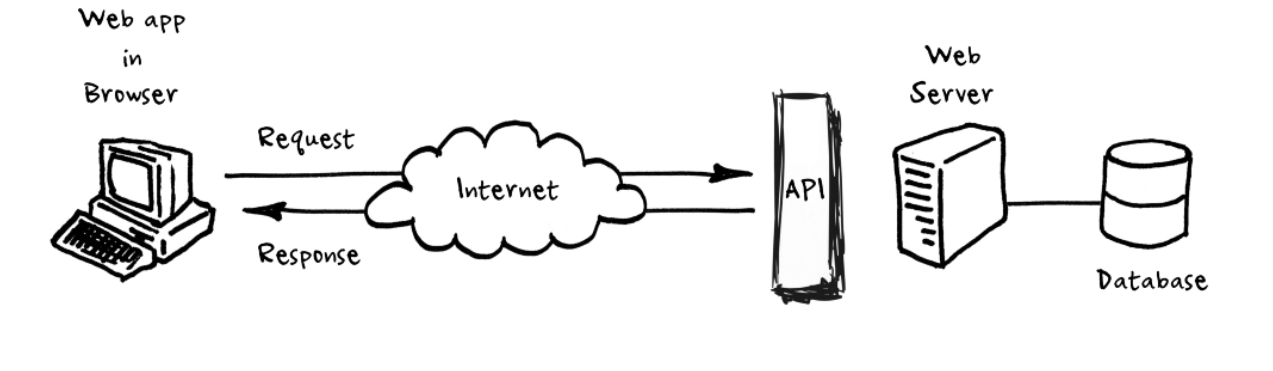
-> The developer creates the API on the server and allows the client to talk to it.

**Def** : A server that creates a interface for interacting with some data. (in context of server)

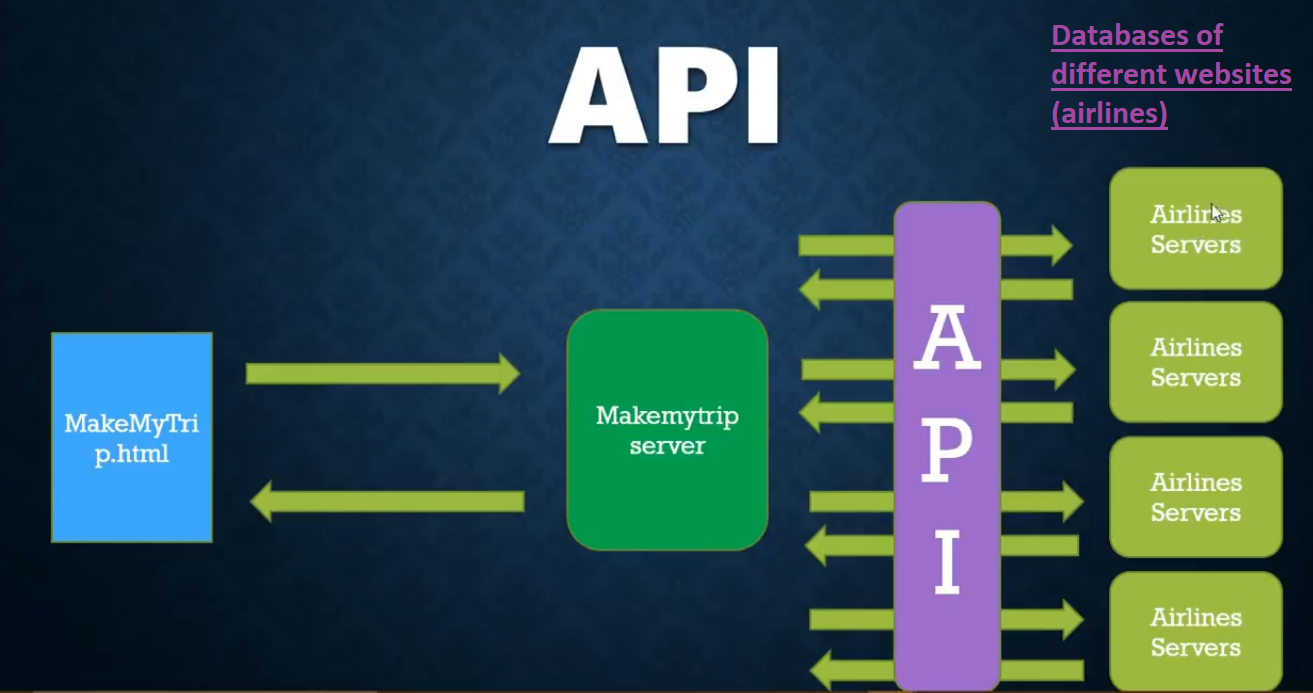
Note : An API is not a database. It is an access point to an app that can access a database.

**API working**

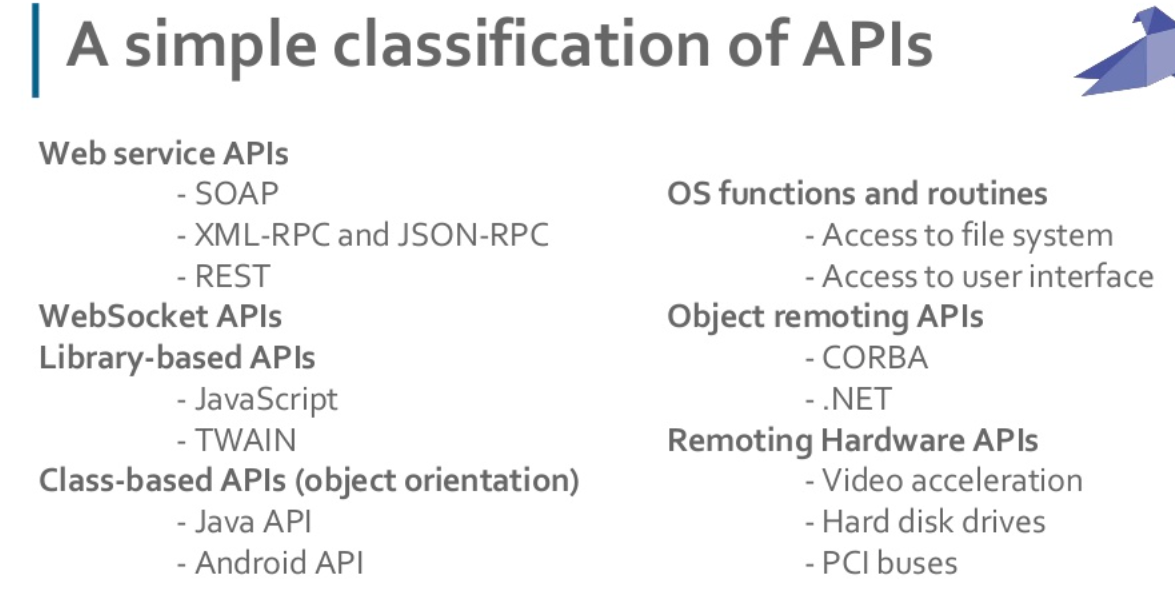




**Real Application example**



**Link :** [**https://www.youtube.com/watch?v=E0Qqpn8ymko&t=173s**](https://www.youtube.com/watch?v=E0Qqpn8ymko&t=173s)



**What is REST API (or Restful API)**

It stands for “Representational State Transfer”.

It is most popular api design pattern.

REST determines how the API looks like.

It is a set of rules that developers follow when they create their API.

**Rules** : <https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/>

Note : stateless (client do not need to know about the server and server donot need to know about the client)

**Terms**

Each URL is called a request while the data sent back to you is called a response.

It’s important to know that a request is made up of four things:

1.) The endpoint

2.) The method

3.) The headers

4.) The data (or body)

**1.) endpoint** : (or route) is the url you request for. It follows this structure:

**2.) The Method**

The method is the type of request you send to the server. You can choose from these five types below:

(i) GET

(ii) POST

(iii) PUT

(iv) PATCH

(v) DELETE

**3.) The Headers**

Headers are used to provide information to both the client and server. It can be used for many purposes, such as authentication and providing information about the body content.

**4.) The Data (Or “Body”)**

The data (sometimes called “body” or “message”) contains information you want to be sent to the server. This option is only used with POST, PUT, PATCH or DELETE requests.

**Nodejs**

Node.js is a javascript running on the server.

Node.js is asynchronous (i.e. does not wait)

A Node. js application runs on single thread and the event loop also runs on the same thread.

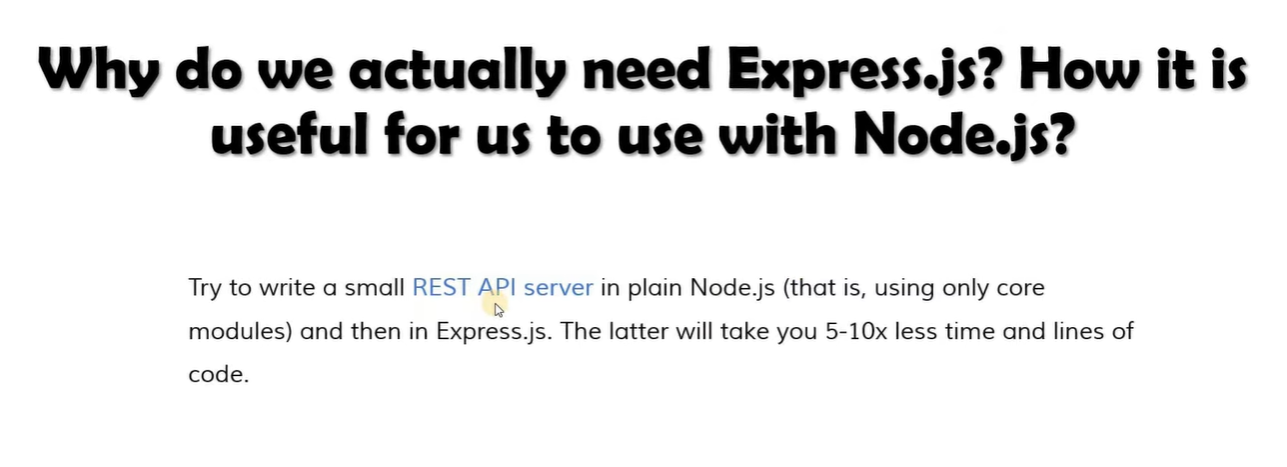
**Link**  <https://medium.com/better-programming/is-node-js-really-single-threaded-7ea59bcc8d64#:~:text=A%20Node.,runs%20on%20the%20same%20thread>.

**Expressjs**

Express.js is a server-side web framework for Node.js.

Why use Express

* Fast and Less lines of code.



// Creating express app

// --------------------

// Import the module

const express = require("express");

// Executing the function stored in express variable And storing the result into 'app' variable

// This statement creates a new express application for you.

const app = express();

// Note : Now this 'app' object have bunch of properties and methods

// Important methods available

// 1.) get - read

// 2.) post - create

// 3.) put - update

// 4.) delete - delete

**Terms**

**1.) Routing :-** Routing refers to how an application’s endpoints (URIs) respond to client requests.

**Routing with nodejs**

const http = require("http");

const fs = require("fs");

// call the createServer() method : This method is used to create server

const server = http.createServer(function (req, res)

{

    //request

    console.log(`Requested page = ${req.url}`);         // We can use url property of request object to get the url from where the request was made

    // Routing

    if(req.url == "/" || req.url== "/home")

    {

        res.write("Hello from the home sides");

        res.end();

    }

    else if(req.url == "/signup" || req.url == "/login")

    {

        res.write("Hello from the signup/login sides");

        res.end();

    }

    else if(req.url == "/about")

    {

        res.write("Hello from the aboutus sides");

        res.end();

    }

    else if(req.url  == "/userapi")

    {

        fs.readFile("userapi.json","utf8", function (err, data)

        {

            console.log(data);          // Print on the terminal

            res.write(data);            // Print on the webpage

            // If we want to access properties

            // We have to convert json into javascript object

            const objData = JSON.parse(data);

            // res.writeHead(200, {"Content-type" : "application/json"});

            res.write(objData[0].name);

            res.end();

        });

    }

    else

    {

        //Message (response) sent to the request made by the client

        res.writeHead(404, {"Content-type" : "text/html"});

        res.write("<h1> 404 error. Page doesn't exist </h1>");

        // The end() function is used to end the response process.

        res.end();

    }

});

// listen() : The listen() method is used to listen the request

// syntax : server.listen(port, hostname, backlog, callback);

server.listen(8000, "127.0.0.1", function ()

{

    console.log("Listening to the port no 8000");

});

// HTTP response status codes

// ------------

// Informational responses (100–199)

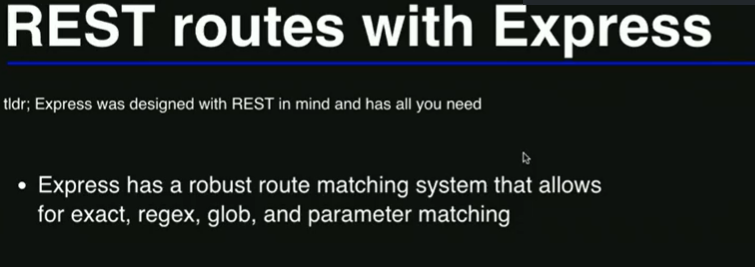
// Successful responses (200–299)

// Redirects (300–399)

// Client errors (400–499)

// Server errors (500–599)

**Routing with expressjs**



**Example**

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

res.send({message : "hello"});

})

**2.) controllers**

**Example**

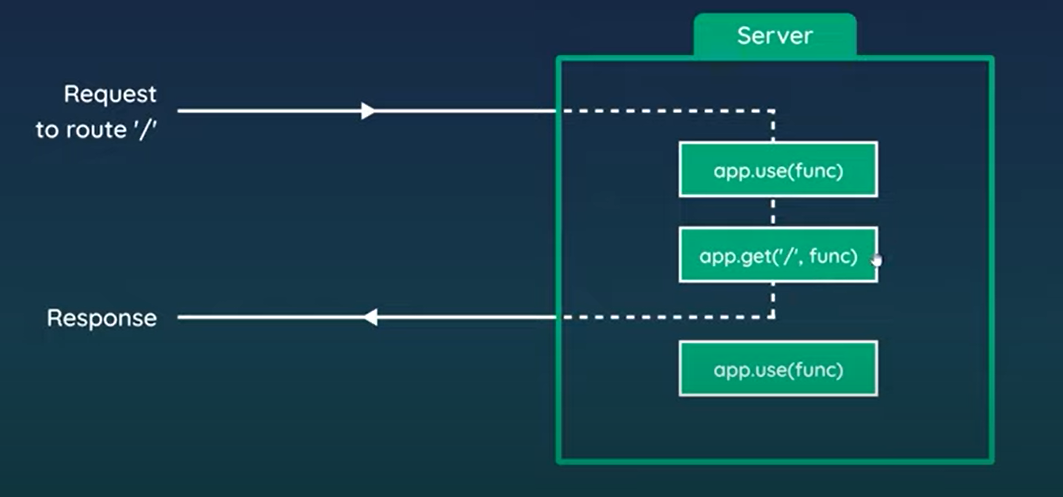
(req,res) => {

res.send({message : "hello"});

}

**3.) Midddleware**

**-> Trigger between request and response**

****

**Note** : Order is very imp in middleware

**Def : List of functions that execute , in order, before your controller.**

**Real example** : where we can use

create your own middleware and check wheather the person is logged in or not from that (i.e. their is no need to call the api if user is not logged in)

**Example (middleware provided by express)**

// Take the permission from express for json data (postman)

app.use(express.json());

// Use bodyParser

app.use(bodyParser.urlencoded({ extended: true }));

// to avoid cors header error

// when website and server is on different domain browser not allow the client to interact with the server because of cross domain.

// It allow the api to be used by other domains

app.use(cors());

**Example (custom middleware)**

// Custom middleware

const log = ((req, res, next) => {

console.log("logging");

next();

})

const express = require("express");

const app = express();

// If we want to run the middleware for all the request

// app.use(log);

**Mongoose**

**Connection nodejs with Mongodb (Atlas)**

// Import mongoose

const mongoose = require("mongoose"); // with mongoose object we will connect to database

// Create connection (connect() returns promises)

// Note : Three stages in promises (i) pending (ii) resolve (iii) reject

mongoose.connect("mongodb+srv://bhupinder:sandeepB0@@cluster.miexj.mongodb.net/phonebook?retryWrites=true&w=majority", { useNewUrlParser: true, useUnifiedTopology : true, useCreateIndex : true})

.then( () => console.log("Connection successfull ......."))

.catch( (err) => console.log(err));

**Data Modeling**

**Schema :** Structure of the document (defines the datatype for each field) , default values, validator.

**Example1 :**

const userschema = new mongoose.Schema

({

name : {type : String, required : true, unique : true, default : “abc”},

password : {type : String, required : true},

});

**Example2:**

const userschema = new mongoose.Schema({

    name : {type: String, required : true},

    password : {type : String, required : true},

    socialmedia : {

        facebook : String,

        github : String

    }

});

**Example3 : Nested schema**

const mongoose = require("mongoose");

// Schema

const socialmediaSchema = new mongoose.Schema({

    facebook : {type : String},

    github : {type : String},

    leetcode : {type : String}

})

const userschema = new mongoose.Schema({

    name : {type: String, required : true},

    password : {type : String, required : true},

    socialmedia : [socialmediaSchema]

});

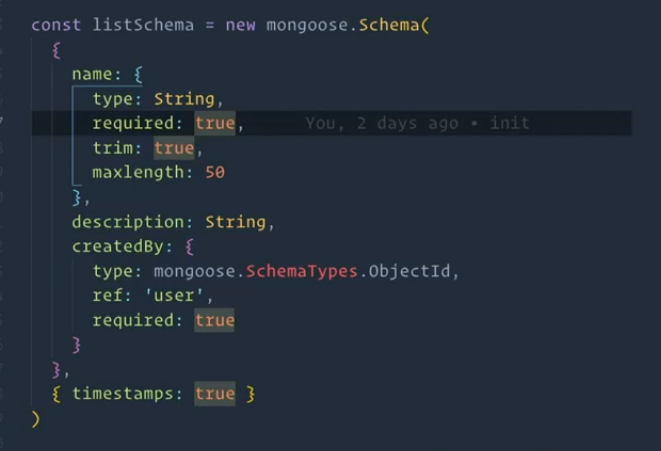
// Model

const User = new mongoose.model("User", userschema);

// Export

module.exports = User;

**Example4 :**

****

**Model :** A model is a wrapper on the Mongoose Schema and Model is used for CRUD operations

**Example (creating model)**

const User = new mongoose.model("user", userschema); // Class (compiling our schema into a Model)

// Note : A model is a class

// Note :

Resources activate the controllers and we have controllers that use the models to query and perform CRUD operations on the models.

**Example** : **Data Model**

const userschema = new mongoose.Schema({

    name : {type: String, required : true},

    password : {type : String, required : true},

    socialmedia : {

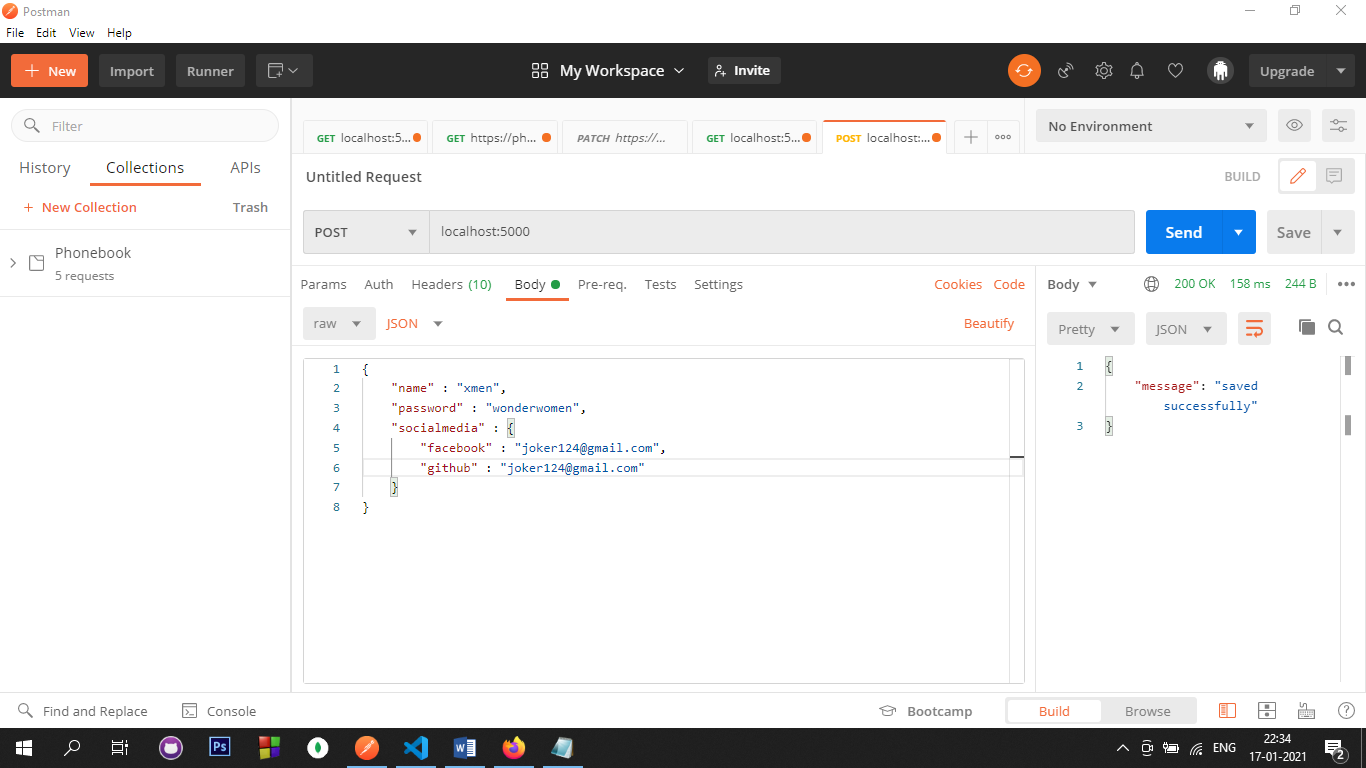
        facebook : String,

        github : String

    }

});

**Sending to server**



**How to insert data in database**

app.post("/", (req,res) => {

    console.log(req.body);

    // Insert in database

    Register.create({name : req.body.name, password : req.body.password, socialmedia : { facebook :  req.body.socialmedia.facebook, github : req.body.socialmedia.github}  }, (err,data) => {

        if(err)

            console.log(err);

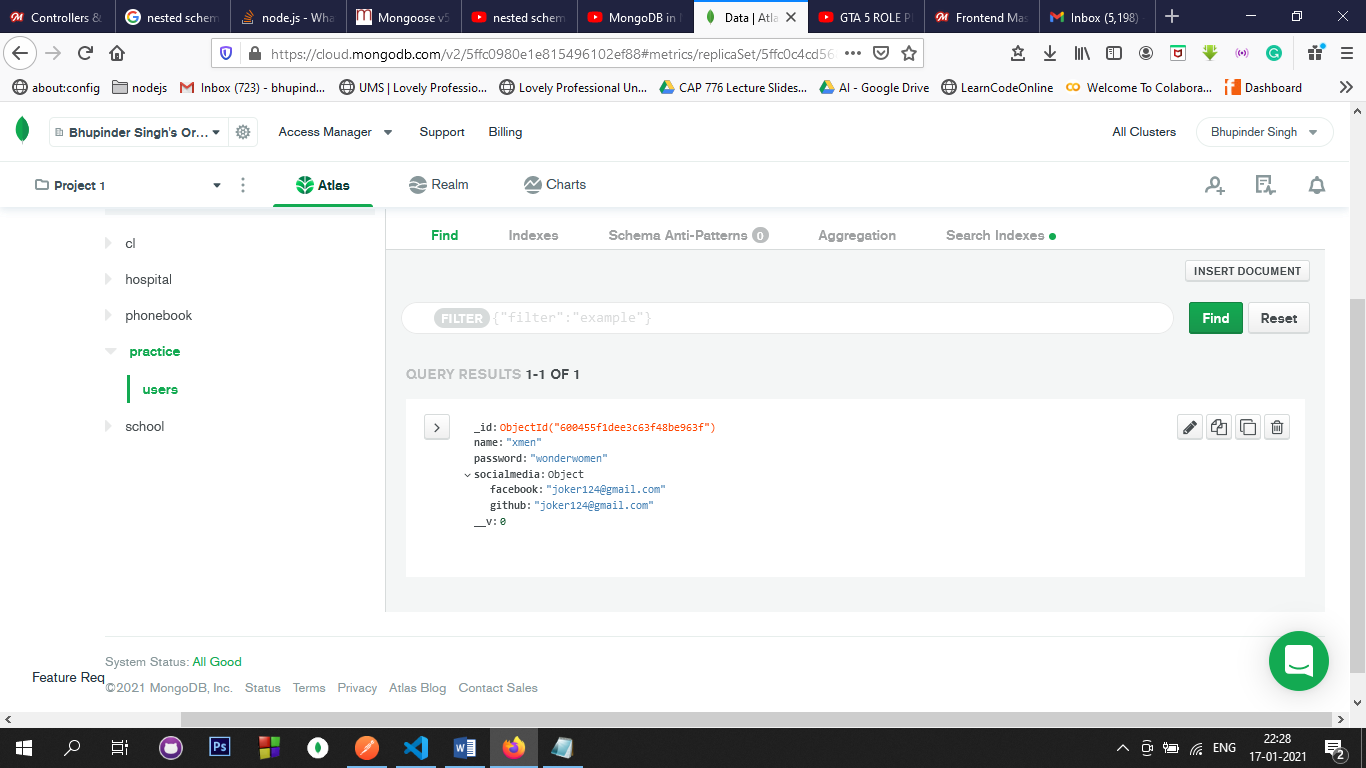
        else

            res.status(200).send({message : "saved successfully"});

    })

})

**Atlas**



**Controllers**

middleware are often reused more than once and often they do not response.

Where as, controller respond and are most of the time specific to one endpoint.

**Note** : In case of error middleware can be used to send response

**Real example**

const express = require("express");

const app = express();

function authenticationMiddleware(req, res, next)

{

  // Check that the user is authenticated using req.headers.Authorization

  // for example

  if (authenticated)

{

    // The user is authenticated, we can go to the next step

    next();

  }

else

{

    // The user is not authenticated, we stop here

    res.status(401);

    res.send("Error during authentication");

  }

}

function getItems(req, res, next)

{

  // Here we focus on the actual response, we assume that the user is  authenticated

  res.send({ items: [] });

}

**Example : Controller**

app.get("/", (req,res) => {

    res.status(200).send({message : "ok get"}); // Controller

})