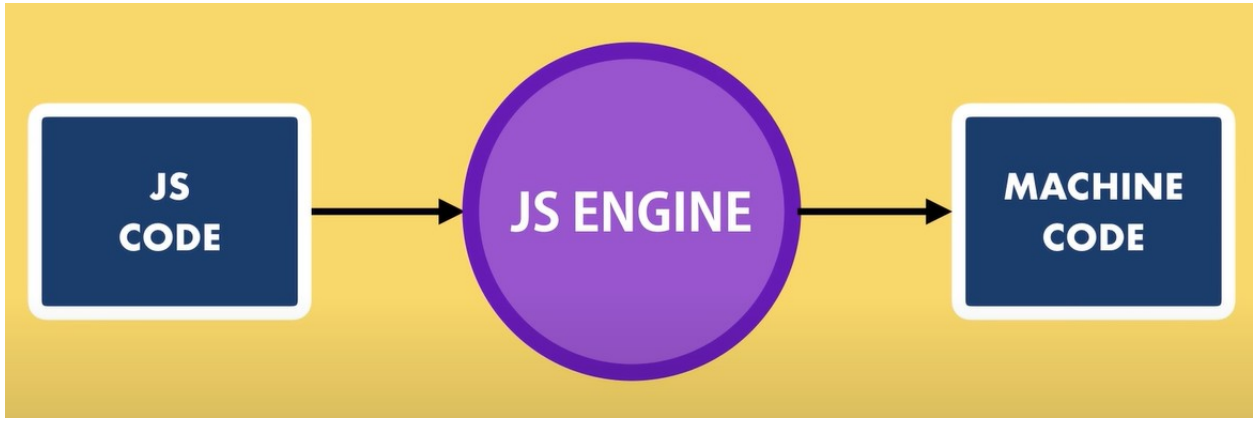
Node.js/Express.js

Node.js Tutorial

- Node.js is an open source server environment.
- It allows us to run JavaScript on the server.
- A **runtime enviroment** for executing JavaScript code outside of browser.
- It runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses asynchronous programming!
- It files have extension **".js".**

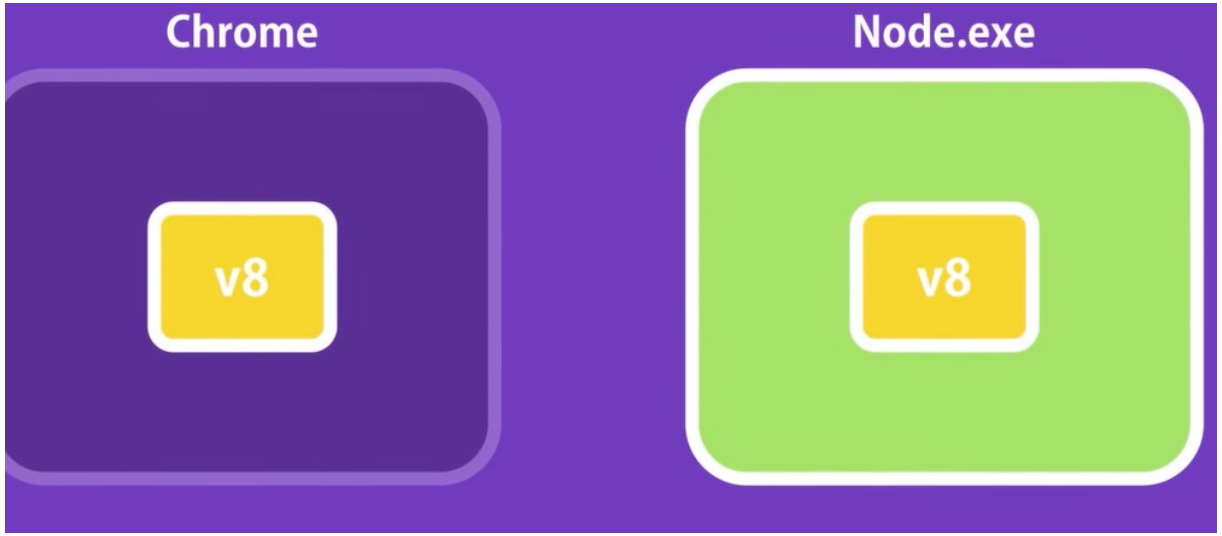
Node Architecture

- Every Browser is JavaScript engine.



Node Architecture

- Combine V8 engine of chrome with C++ programming and run JavasScript outside the browser.
- Note: Node is not a programming langauge and not framework. It is runtime enviroment for executing JavaScript Code outside the browser.
- Developed by Ryan Dahl in 2009.



Blocking / Synchronous Architecture

- A common task for a web server can be to open a file on the server and return the content to the client.
- Here is how **PHP** or **ASP** handles a file request:

1. Sends the task to the computer's file system.
2. **Waits** while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Non-blocking / Asynchronous Architecture

- Here is how **Node.js** handles a **file request**:

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

- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs **single-threaded**, non-blocking, asynchronous programming, which is very memory efficient.

Capabilities of Node.js

- Node.js can generate dynamic page content.
- Node.js can **create, open, read, write, delete, and close files** on the server.
- Node.js can **collect form data.**
- Node.js can **add, delete, modify data in your database.**

Working with Node.js

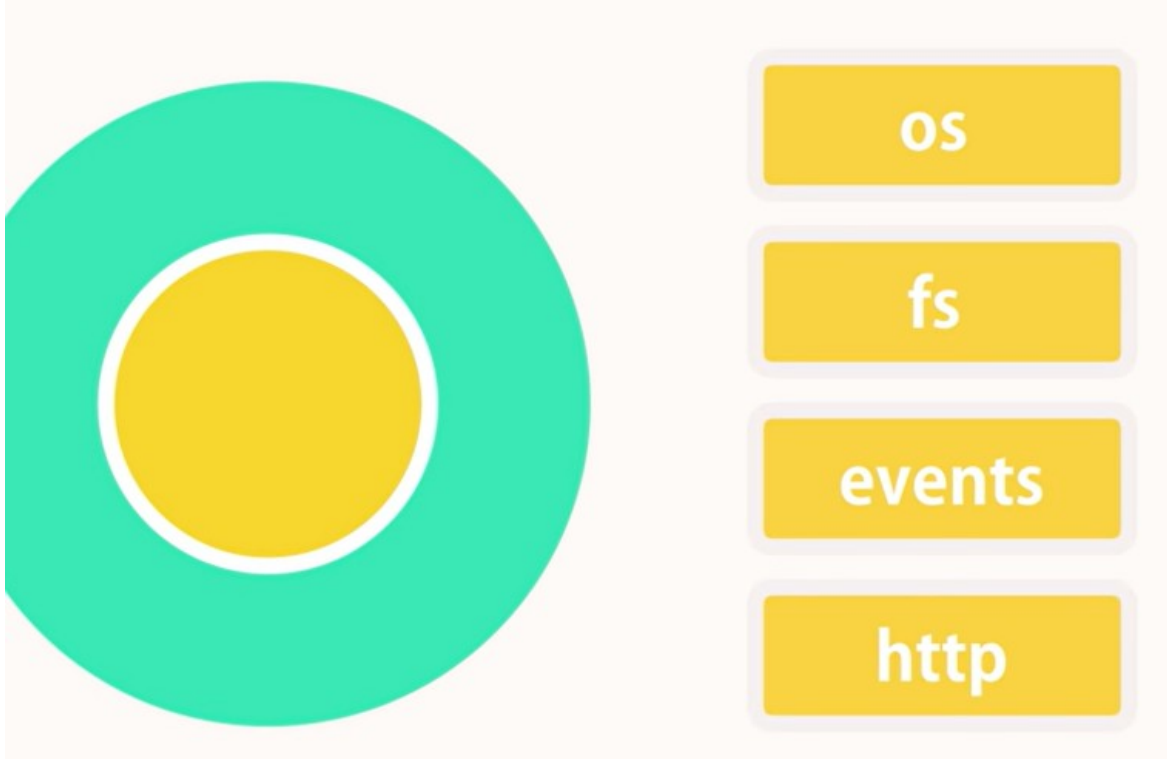
- Create a dirctory.
  - mkdir NodeApp
  - create a file using an editor and save as **app.js**
  - Write a function given below and run in cmd **node app.js**

```
function sayHello(name)
{
    console.log('Hello '+ name);
}
sayHello("Ali");
```

```
cation>node app.js
```

Node Module System

- Consider modules to be the same as JavaScript libraries.



Node Module System

- Every node application has **atleast one file** or one module that is app.js.
- **Every file is a module**, and the functions and variables defined in that module are only accessible within that module.
- console.log(module);

```
{
  id: '.',
  path: 'D:\\FAST Peshwar\\Spring-2024\\Web Technology\\Example\\NodeApplication',
  exports: {},
  filename: 'D:\\FAST Peshwar\\Spring-2024\\Web Technology\\Example\\NodeApplication\\app.js',
  loaded: false,
  children: [],
  paths: [
    'D:\\FAST Peshwar\\Spring-2024\\Web Technology\\Example\\NodeApplication\\node_modules',
    'D:\\FAST Peshwar\\Spring-2024\\Web Technology\\Example\\node_modules',
    'D:\\FAST Peshwar\\Spring-2024\\Web Technology\\node_modules',
    'D:\\FAST Peshwar\\Spring-2024\\node_modules',
    'D:\\FAST Peshwar\\node_modules',
    'D:\\node_modules'
  ]
}
```

Creating Module

- Now add a new module to the existing application.
- logger.js

```
var url = 'http://mylogger.com/ ';
Comment Code
function log(message) {
        // send an HTTP request
        console.log(message);
}
```

- To Make it **public**. we need to export it.
- Use the exports keyword to make properties and methods available outside the module file.

```
module.exports.log = log;
```

Loading Module

- To load module, use require() function. It takes one argument.
- Notice that we use **./** to locate the module, that means that the module is located in the same folder as the Node.js file.

```
require('./logger.js');    OR require('./logger');
```

```
var logger = require('./logger.js');
console.log(logger);
```

```
{ log: [Function: log] }
```

```
logger.log('My Name is Ali');
```

Exporting more than one Modules

```
function log2(message) {
    console.log(message);
}
Comment Code
function anotherFunction() {
    console.log('Another function');
}
```

```
module.exports = {
    log: log2,
    anotherFunction: anotherFunction
};
```

- Loading in app.js

```
const logger = require('./logger.js');
logger.log('My Name is Ali');
logger.anotherFunction();
```

Built-in Modules

- Visit https://nodejs.org/ and click on "DOCS"; you will find many useful built-in modules.
- Such as HTTP, FileSystem, Path
- First we need to call the module using require function.
- Example is given below using path module

```
const path = require('node:path');
const pathobj = path.parse(__filename);
console.log(pathobj);
```

```
{
  root: 'D:\\',
  dir: 'D:\\FAST Peshwar\\Spring-2024\\Web Technology\\Example\\NodeApplication',
  base: 'app.js',
  ext: '.js',
  name: 'app'
}
```

Built-in Modules

- OS built-in module Example

```
const os = require('os');

var totalMemory = os.totalmem();
var freeMemory = os.freemem();

console.log('Total Memory: ' + totalMemory);
```

```
// Template string
// ES6 / ES2015 : ECMAScript 6

console.log(`Total Memory: ${totalMemory}`);
```

Built-in File System Modules

```
const fs = require('node:fs');
```

```
const files = fs.readdirSync('./');
console.log(files);
```

- The above readdirSync is Synchronous call.
- It returns array of strings

```
[
  'app.js',
  'logger.js',
  'node_modules',
  'package-lock.json',
  'package.json'
]
```

Aysnchronous method calls

```
const fs = require('node:fs');

fs.readdir('./', function (err, files) {
    if(err) console.log('Error ',err);
    else console.log('Result', files);
});
```

```
Result [
  'app.js',
  'logger.js',
  'node_modules',
  'package-lock.json',
  'package.json'
]
```

- All Ansychronous methods will take a function as a last argument.
- Node will call this method when the Ayschronous operation is completed.
- This function is called callback function.

Error in Aysnchronous method calls

```
const fs = require('node:fs');
```

```
fs.readdir('$', function (err, files) {
    if(err) console.log('Error ',err);
    else console.log('Result', files);
});
```

```
Error  [Error: ENOENT: no such file or directory, scandir 'D:\FAST Peshwar\Spring-202
ation\$'] {
  errno: -4058,
  code: 'ENOENT',
  syscall: 'scandir',
  path: 'D:\\FAST Peshwar\\Spring-2024\\Web Technology\\Example\\NodeApplication\\$'
}
```

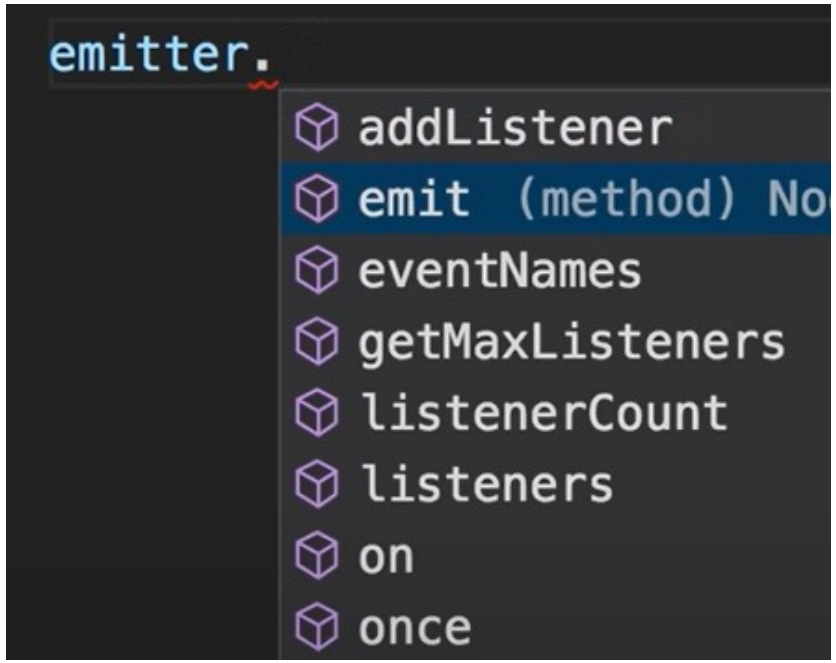
Event in Node

```
const EventEmitter = require('node:events');
```

- Every action on a computer is an event. Like when a file is opened.
- EventEmitter is class. First creat instance of class, i.e., emitter.

```
const emitter = new EventEmitter();
```

- EventEmitter class has methods.



Use emit method of Event Class

```
emitter.emit();
```

- To fire an event, use the emit() method. emit() raise an Event.
- Emit mean make a noise or produce something - signaling.

```
// Register a Listener
emitter.on('CallMessage', function(){
    console.log('Event Message received');
});
```

- Nothing is going to happen. We need to register the event.

```
emitter.emit("CallMessage");
```

Event Arguments

```
// event Arguments
emitter.on('CallMessage', (arg)=>{
    console.log('Event Message received', arg);
});

emitter.emit("CallMessage",{id:2,url:'http://'});
```

Extending Event

- Logger.js file

```
class Logger extends EventEmitter{
    // Inside we do not write the function keywoed
    log(message) {
        console.log(message);
        // Raise event and inside the class use this.e
        this.emit("CallMessage",{id:2,url:'http://'});
    }
}
```

- App.js file

```
const Logger = require('./logger');
const logger = new Logger();

// Register a listener
logger.on('messageLogged', (arg) => {
  console.log('Listener called', arg);
});

logger.log('message');
```

URL Module

- The URL module splits up a web address into readable parts.
- Parse an address with the url.parse() method, and it will return a URL object.

```
const url = require('url');
var add = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(add, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

const qdata = q.query; //returns an object: { year: 2017, month: 'febru
console.log(qdata.month); //returns 'february'
```

HTTP Module (Creating http server)

```
const http = require('http');
const server = http.createServer();

// Register listener
server.on('connection',()=>{
    console.log('I am listening on on port 3000');
});

// call an event
server.listen(3000);
```

```
I am listening on on port 3000
```

HTTP Module (Creating http server)

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

- Use the createServer() method to create an HTTP server.

- The function passed into the http.createServer() method, will be executed when someone tries to access the localhost on port 3000.

HTTP Server with arguments

```
const server = http.createServer((req, res)=>{
    if(req.url==='/'){
        // Write a response to the client
        res.write('Hello World! \n');
        res.write(req.url); // print URL
        res.end(); // End the response
    }
    if(req.url==='/api/courses'){
        res.write(JSON.stringify([1, 2, 3]));
        res.end();
    }
});

// The server object listen on port 3000
server.listen(3000);
```


```
Hello World!
/
```


```
[1,2,3]
```

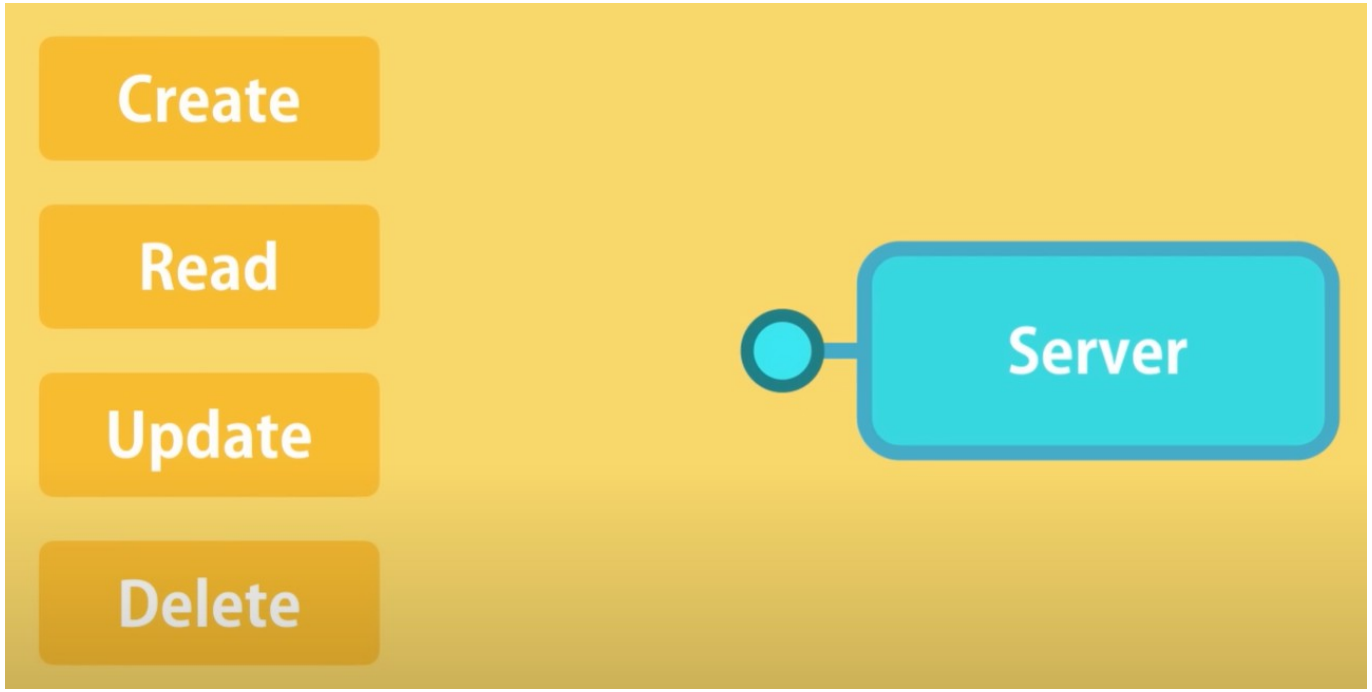
Express

```
const server = http.createServer((req, res)=>{
    if(req.url==='/'){
        // Write a response to the client
        res.write('Hello World! \n');
        res.write(req.url); // print URL
        res.end(); // End the response
    }
    if(req.url==='/api/courses'){
        res.write(JSON.stringify([1, 2, 3]));
        res.end();
    }
});
```

- The above approach is not good for large and complex application.
- If we have too many URL, we need to hard code each of them.
- Use Express, which is light weight framework

RESTful services or APIs

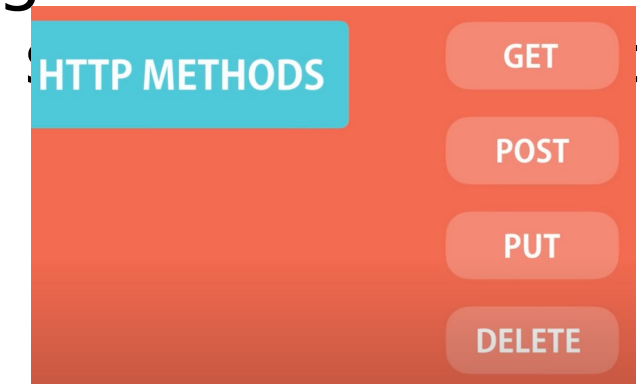
- We create service that provide the following operations.
- This operations all togather are refered to CRUD Operations.




```
http://vidly.com/api/customers
```

HTTP Methods

- GET: for getting data: Requests using GET should only retrieve data and should have no other effec
- POST: creating data: The POST method requests that the server accept the data enclosed in the request.
- PUT: for updating data: The PUT method requests that the server accept the data enclosed in the request as a modification to existing object identified by the URI
- DELETE: for deleting data: DELETE method requests that the server delete the s e



GET Methods Example

Request

```
GET /api/customers
```

Response

```
[
    { id: 1, name: '' },
    { id: 2, name: '' },
    …
]
```

Request

```
GET /api/customers/1
```

PUT (UPDATE) and DELETE Methods Example

Request

```
PUT /api/customers/1

{ name: '' }
```

Response

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

Request

```
DELETE /api/customers/1
```

POST Methods Example

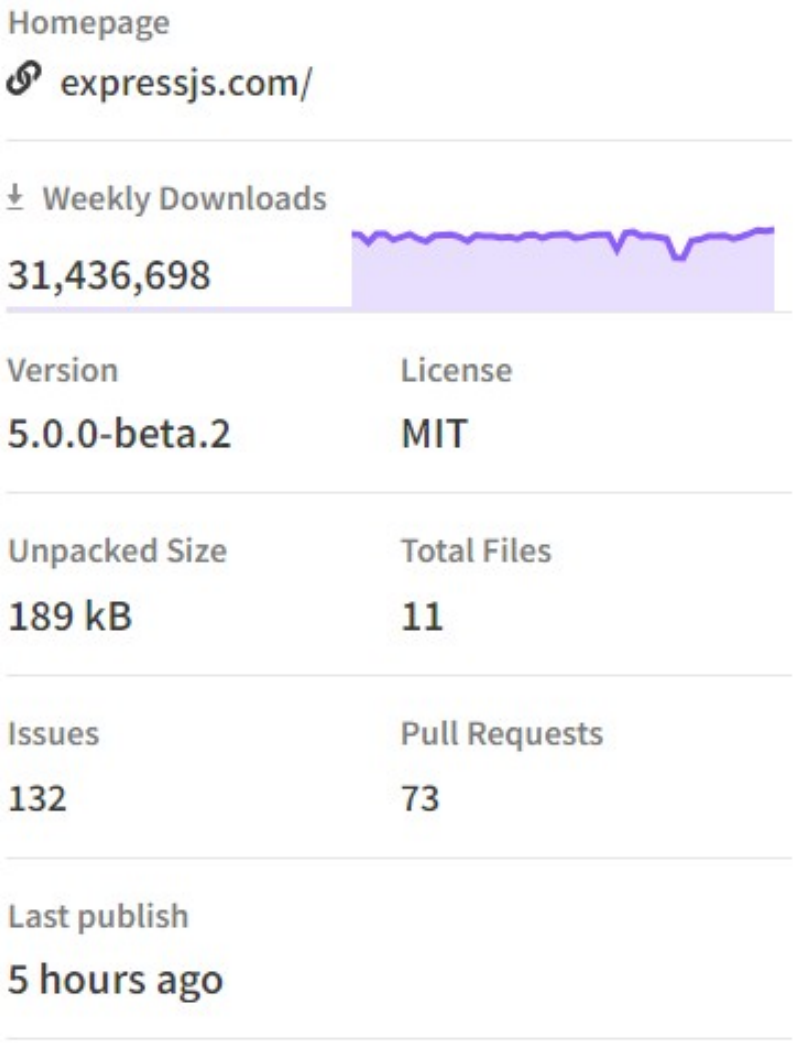
Request

```
POST /api/customers

{ name: '' }
```

```
GET /api/customers
GET /api/customers/1
PUT /api/customers/1
DELETE /api/customers/1
POST /api/customers
```

Introducing Express

- ExpressJS is a web application framework.
- It provides us with a simple API to build websites web apps and back ends
- Express was developed by TJ Hol
- It is maintained by the Node.js fo
- go to https://www.npmjs.com/
- Search express in Search Bar



Express Installation

- make a directory and run the command inside dir

```
npm init
```

-

```
$npm i express
```

```
Wrote to D:\FAST Peshwar\Spring-2024\Web Technology\Example\expressA\package.json:

{
  "name": "expressa",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Working with Express

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


app.get()
app.post()
app.put()
app.delete()
```

First Web Server

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

app.get('/', (req, res) => {
  res.send('Hello World');
});

app.get('/api/courses', (req, res) => {
  res.send([1, 2, 3]);
});

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

app.get(route, callback)

Nodemon

```
$npm i -g nodemon
```

```
nodemon index.js
```

```
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Listening on port 3000 ...
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Listening on port 3000 ...
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Listening on port 3000 ...
```

Evironmental Variable

- Port is dynamically assigned by the hosting provider
- port number can change every time our application or service restarts or is redeployed.
- 3000 port may work on local but not on server. We need to set Environment Variable.
- Environment variables are variables that are defined in the environment in which the Node.js process is running.
- Express.js allows us to access environment variables using the **process.env** object

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

```
app.listen(3000, ()=>console.log(`Listening on port ${port} `));
```

```
>SET PORT=5000 && node index.js
```

Router Parameters

- We can pass parameters to the app.get() route handler using route parameters or query parameters.
- Route parameters are part of the URL path and are defined by placing a colon (:) followed by the parameter name in the route path

```
// /api/courses/1
app.get('/api/courses/:courseID', (req, res)=>{
const CourseID = req.params.courseID;
res.send(`Your Response is for Course Id ${CourseID}`);
});
```

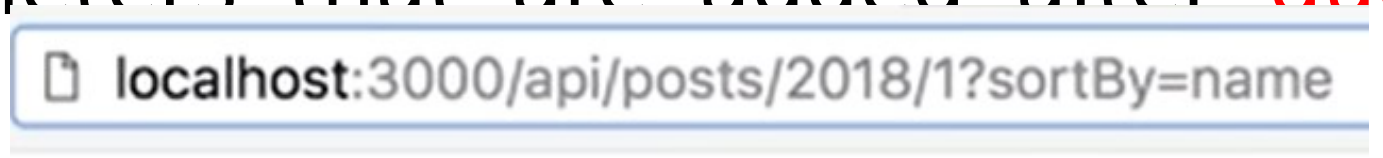
Router Parameters

- We can pass multiple parameters to the app.get() route handler using route parameters or query parameters

```
app.get('/api/lastlogin/:year/:month',(req,res)=>{
    //res.send(req.params);
    const year = req.params.year;
    const month = req.params.month;
```

Router Parameters

- We can pass query string as parameters to the app.get().
- Parameters that are added after question mark (?).




```
app.get('/api/lastlogin/:year/:month',(req,res)=>{
        res.send(req.query);

});
```

Pattern Matched Routes

- We can also use **regex** to restrict URL parameter matching.
- Let us assume we need the id to be a 5-digit long n

```
app.get('/things/:id([0-9]{5})', function(req, res){
    res.send('id: ' + req.params.id);
});
```

Pattern Matched Routes

- To display a user defined message, when the Regular Expression does not match or URL route is defined.
- Use * in the route.
- Important – This should be placed after all your routes, as Express matches routes from start to end of the in

```
//Other routes here
app.get('*', function(req, res){
   res.send('Sorry, this is an invalid URL.');
});
app.listen(3000);
```

Handling GET Request

- Retrieve specific data using app.get().

```
const courses=[
    {id:1,name:"Data Structure"},
    {id:2,name:"Machine Learning"},
    {id:3,name:"Web Technology"}
];
```

```
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");
res.send(course.name);
```

Handling POST Request

```
// Add for the post request JSON middle ware
app.use(express.json());
let courses=[
    {id:1,name:"Data Structure"},
    {id:2,name:"Machine Learning"},
    {id:3,name:"Web Technology"}
];
```

BODY

```
{
  "name":"Programming Fundamental"
}
```

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

- **Json()** is a middleware function designed to handle the data inside the body of a POST request.
- Middleware functions have access to the **request object** (**req**), the **response object** (res)

Input Validation

```
if(!req.params.name || req.params.name<3){
    // 400 bad request
    req.status(400).send("Name is Required
    return;
}
```

```
// console.log(result.error.details[0].message);
 if(result.error){
    res.status(400).send(result.error);
    return;
 }
```

```
const Joi = require('joi'); // It returns the class.
```

```
  const schema = Joi.object(
      {
      name:Joi.string().min(3).required()
  });
  const result = schema.validate(req.body);
  console.log(result);
```

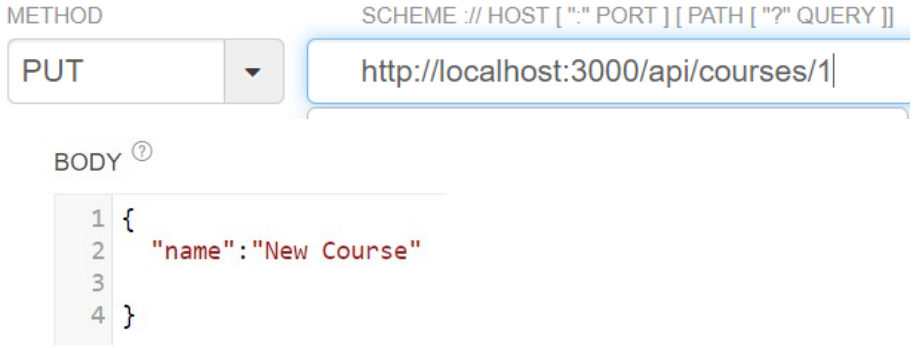
```
{
  value: { name: '' },
  error: [Error [ValidationError]: "name" is not allowed to be empty] {
    _original: { name: '' },
    details: [ [Object] ]
  }
}
```

Handling PUT Request

```
// Handling PUT Request
app.put('/api/courses/:id',(req,res)=>{
// Look up the course
// if not exist, return 404

// if exist validate the request
// if invalid, return 400, Bad request

//update course
// return the updated course

});
```

```
const course = courses.find(c => c.id===parseInt(req.params.id));
if(!course) res.status(404).send("The Course is not found");
```

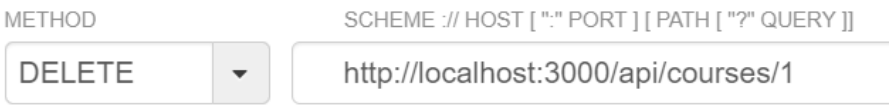
```
const schema = Joi.object({
    name:Joi.string().min(3).required()
});
const result = schema.validate(req.body);
if(result.error){
    res.status(400).send(result.error.details[0].message);
    return;
}
```




```
course.name = req.body.name;
res.send(course);
```

Handling DELETE Request

```
app.delete('/api/courses/:id',(req,res)=>{
    // Look up the course
    // Not Exist, Return 404
    const course = courses.find(c => c.id===parseInt(req.params.id));
    if(!course) res.status(404).send("The Course is not found");

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

    // Return same course
    res.send(course);
});
```



Routers

- Defining routes in single file is very tedious to maintain.
- To separate the routes from our main index.js file, we use Express.Router().
- Create a new file called things.
- Then use in **index.js**

```
var express = require('Express');
var app = express();

var things = require('./things.js');

//both index.js and things.js should be in same directory
app.use('/things', things);

app.listen(3000);
```

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res){
   res.send('GET route on things.');
});
router.post('/', function(req, res){
   res.send('POST route on things.');
});

//export this router to use in our index.js
module.exports = router;
```

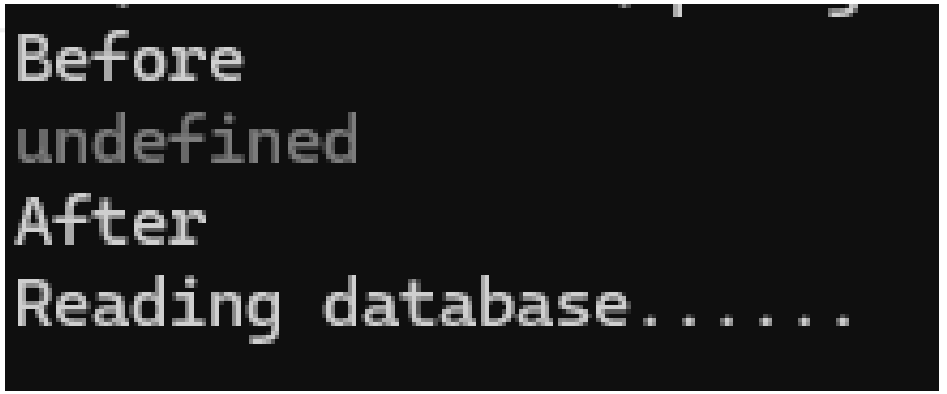
Asynchronous JavaScript Demo

```
console.log('Before');
setTimeout(()=>{
    console.log('Reading database.....')
},2000);
console.log('After');
```

```
Before
After
Reading database.....
```

Asynchronous JavaScript Demo

```
console.log('Before');
const user = getUser(2);
console.log(user);
console.log('After');
function getUser(id)
{
    setTimeout(() => {
    console.log('Reading database.....');
    return {id:id,name:"Ali"};
    }, 2000);
}
```



Asynchronous JavaScript Demo

```
console.log('Before');
getUser(2, function(user){
    console.log('User ', user);
});
console.log('After');
function getUser(id,callback)
{
    setTimeout(() => {
    console.log('Reading database.....');
    callback({id:id,name:"Ali"});
    }, 2000);
}
```

- Any Question