**NODEJS**

### NPM: Node Package Manager

While we are trying to understand the basics of Node.js, we definitely don’t want to miss out on its built-in support for package management using npm. A popular package library – npm is the most prized possession of Node.js community. It contains millions of downloadable libraries according to the specific requirement. These massive libraries are absolutely free of cost with its registry. With each passing day, these libraries are getting bigger rapidly, making the Node.js community stronger.

Open-source developers from every continent use npm to share or borrow packages, and many enterprises utilize them for their private developments too.

### Top Useful npm Modules

The minimalist approach inspires some of the most useful npm modules today.

# NPM: Express.js                            # NPM: Sails.js  
# NPM: Meteor.js                             # NPM: Hapi.js  
# NPM: Koa.js                                   # NPM: Total.js  
# NPM:Nest.js                                   # NPM: Loopback.js  
# NPM:Mongo.js                              # NPM: socket.js

These useful npm nodules help you create a module that allows focusing on many essential elements such as high performance, automate time-consuming tasks, robust routing, and super-high test coverage.

**Modules**

In Node.js, **Modules** are the blocks of encapsulated code that communicates with an external application on the basis of their related functionality.

Modules can be a single file or a collection of multiples files/folders.

The reason programmers are heavily reliant on modules is because of their re-usability as well as the ability to break down a complex piece of code into manageable chunks.

**Modules are of three types:**

* Core Modules
* local Modules
* Third-party Modules

**Core Modules:** Node.js has many built-in modules that are part of the platform and comes with Node.js installation. These modules can be loaded into the program by using the **require** function.

Loading Core Modules

In order to use Node.js core or NPM modules, you first need to import it using require() function as shown below.

**Syntax:**

var module = require('module\_name');

| Core Module | Description |
| --- | --- |
| http | http module includes classes, methods and events to create Node.js http server. |
| [url](https://nodejs.org/api/url.html) | url module includes methods for URL resolution and parsing. |
| [querystring](https://nodejs.org/api/querystring.html) | querystring module includes methods to deal with query string. |
| path | path module includes methods to deal with file paths. |
| [fs](https://nodejs.org/api/fs.html) | fs module includes classes, methods, and events to work with file I/O. |
| [util](https://nodejs.org/api/util.html) | util module includes utility functions useful for programmers. |

Example of http module to create a web server

var http = require('http');

var server = http.createServer(function(req, res){

//write code here

});

server.listen(5000);

**Local Modules:** Unlike built-in and external modules, local modules are created locally in your Node.js application.

These modules include different functionalities of your application in separate files and folders. You can also package it and distribute it via NPM, so that Node.js community can use it.

For example, if you need to connect to sql server / MongoDB and fetch data then you can create a module for it, which can be reused in your application.

**Third-party modules:** Third-party modules are modules that are available online using the Node Package Manager(NPM). These modules can be installed in the project folder or globally. Some of the popular third-party modules are mongoose, express, angular, and react.

**Example:**

* npm install express
* npm install mongoose
* npm install -g @angular/cli

**ES6 MODULES**

Since 2015, Node.js has been embracing JS ES6 along with the new changes added every year graciously.

In ES6 Module prominent keywords included

import

export

These keywords introduced to handle working across the packages and the files in NodeJS.

Both the terms are critical in Node.js.

However, it is expected to become standard eventually.

#### ****A Little About ES6 Module: What it is and Why Should You Use It with Node.js?****

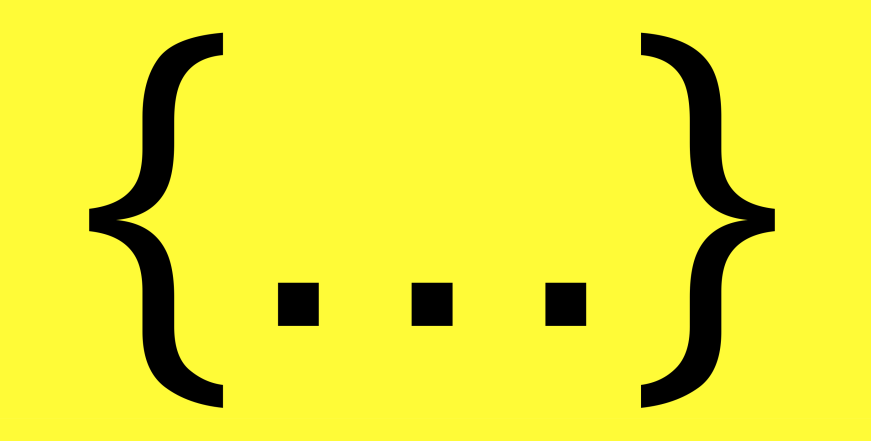
An ES6 module is a file containing JS code, an automatically strict-mode code.

And, you can use import and export in modules.

**Some prominent features of the ES6 module:**  
1. **Arrow function**  
In ES6 module functions, you don’t have to use curly brackets or type the function keyword. Instead, they use a new token ‘=>.’  
Syntax: (parameters)=>{Statements}

Arrow functions are majorly used in:

* Return number function
* Return Array function
* Return number function with parameter
* Return object function



**2. Template Literal (`)**  
It provides an alternative syntax to define strings called Template Literals.

This syntax utilizes backticks (` `) rather than regular quotes or double quotes as delimiters.

The prominent advantage of this feature is that it can interpolate variables or expressions using ${expression} inside the string.

`${...}`

Top of Form

Bottom of Form

**3. Module Exports and Imports**

* ES6 syntax is more readable, and it comes up with keyword export default.
* So, when you want to export a module by default and import the entire module in the source file, this module will do that without hassle.
* You can even import and export multiple variables from a single module using ES6.

For example, if you export:  
Export const a=1  
Export const b=2  
Then, you can import it all together, like:  
import {a,b} from testModule

**Import –** The keyword ‘import’ works with objects and allows destructuring assignment when importing values. As a result, you can create smaller projects by just including specific values—the import keyword functions in connection with the export keyword.

**Export –** Export also works with objects, and when working with multiple objects, it creates exported objects created with inputs.

Functions:

**Function with name**

function test(){

}

test()

Anonymous Funciton: Functions which are not having any name

var person=function(){

}

person()

**Arrow function:**

Var users=()=>console.log(“ “);

Users()

--------------------------------------------------------------

**Ecma6 Var , let, const, templating:**

* var declarations are globally scoped or function scoped while let and const are block scoped.
* var variables can be updated and re-declared within its scope;
* let variables can be updated but not re-declared;
* const variables can neither be updated nor re-declared.

---------------------------------------------------------------------------

**Global objects**

\_ \_dirname

\_ \_filename

require

exports

console

**require() function:**

Node.js follows the CommonJS module system, and the built-in **require** function is the easiest way to include modules that exist in separate files. The basic functionality of require is that it reads a JavaScript file, executes the file, and then proceeds to return the **exports** object.

**http module**==> to create server we use http core module

**server.listen()** ==>The server.listen() is an inbuilt application programming interface of class Socket within module which is used to start the server to listen the encrypted connection.

const server.listen([port[, host[, backlog]]][, callback])

**res.write()**

**res.end()**

**File System Module**

readFile() / appendFile() / unlink() methods

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

To include the File System module, use the require() method: var fs = require('fs');

The **fs.readFile()** method is an inbuilt method which is used to read the file. This method read the entire file into buffer. To load the **fs module** we use require() method.

var http = require('http');  
var fs = require('fs');  
http.createServer(function (req, res) {  
**fs.readFile(hello.txt, function(err, data) {**    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write(data);  
    return res.end();  
  });  
}).listen(8080);

The **fs.appendFile() method** is used to asynchronously append the given data to a file. A new file is created if it does not exist. The options parameter can be used to modify the behavior of the operation.

var fs = require('fs');

fs.appendFile('hello.txt', 'Hello content! appended', function (err) {

  if (err) throw err;

  console.log('Saved!');

});

**unlink() method** is used to delete a file

const fs=require("fs");

fs.unlink(\_\_dirname+"/hello.txt", (err, data)=>{

    if(err) throw err;

    console.log("File deleted"+data)

});

### HTTP response status codes

Code Constant Reason Phrase

100 CONTINUE Continue

101 SWITCHING\_PROTOCOLS Switching Protocols

102 PROCESSING Processing

200 OK OK

201 CREATED Created

202 ACCEPTED Accepted

203 NON\_AUTHORITATIVE\_INFORMATION Non Authoritative Information

204 NO\_CONTENT No Content

205 RESET\_CONTENT Reset Content

206 PARTIAL\_CONTENT Partial Content

207 MULTI\_STATUS Multi-Status

300 MULTIPLE\_CHOICES Multiple Choices

301 MOVED\_PERMANENTLY Moved Permanently

302 MOVED\_TEMPORARILY Moved Temporarily

303 SEE\_OTHER See Other

304 NOT\_MODIFIED Not Modified

305 USE\_PROXY Use Proxy

307 TEMPORARY\_REDIRECT Temporary Redirect

308 PERMANENT\_REDIRECT Permanent Redirect

400 BAD\_REQUEST Bad Request

401 UNAUTHORIZED Unauthorized

402 PAYMENT\_REQUIRED Payment Required

403 FORBIDDEN Forbidden

404 NOT\_FOUND Not Found

405 METHOD\_NOT\_ALLOWED Method Not Allowed

406 NOT\_ACCEPTABLE Not Acceptable

407 PROXY\_AUTHENTICATION\_REQUIRED Proxy Authentication Required

408 REQUEST\_TIMEOUT Request Timeout

409 CONFLICT Conflict

410 GONE Gone

411 LENGTH\_REQUIRED Length Required

412 PRECONDITION\_FAILED Precondition Failed

413 REQUEST\_TOO\_LONG Request Entity Too Large

414 REQUEST\_URI\_TOO\_LONG Request-URI Too Long

415 UNSUPPORTED\_MEDIA\_TYPE Unsupported Media Type

416 REQUESTED\_RANGE\_NOT\_SATISFIABLE Requested Range Not Satisfiable

417 EXPECTATION\_FAILED Expectation Failed

418 IM\_A\_TEAPOT I'm a teapot

419 INSUFFICIENT\_SPACE\_ON\_RESOURCE Insufficient Space on Resource

420 METHOD\_FAILURE Method Failure

422 UNPROCESSABLE\_ENTITY Unprocessable Entity

423 LOCKED Locked

424 FAILED\_DEPENDENCY Failed Dependency

428 PRECONDITION\_REQUIRED Precondition Required

429 TOO\_MANY\_REQUESTS Too Many Requests

431 REQUEST\_HEADER\_FIELDS\_TOO\_LARGE Request Header Fields Too Large

451 UNAVAILABLE\_FOR\_LEGAL\_REASONS Unavailable For Legal Reasons

500 INTERNAL\_SERVER\_ERROR Internal Server Error

501 NOT\_IMPLEMENTED Not Implemented

502 BAD\_GATEWAY Bad Gateway

503 SERVICE\_UNAVAILABLE Service Unavailable

504 GATEWAY\_TIMEOUT Gateway Timeout

505 HTTP\_VERSION\_NOT\_SUPPORTED HTTP Version Not Supported

507 INSUFFICIENT\_STORAGE Insufficient Storage

511 NETWORK\_AUTHENTICATION\_REQUIRED Network Authentication Required

13th August:

**EXPRESSJS**

Fast, minimalist web framework for Node.js

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API (Application Programming Interface) is quick and easy.

# Install Express js

Firstly, you have to install the express framework globally to create web application using Node terminal. Use the following command to install express framework globally.

1. npm install -g express

**GET and POST**

**GET** and POST both are two common HTTP requests used for building REST API's.

**GET** requests are used to send only limited amount of data because data is sent into header while POST requests are used to send large amount of data because data is sent in the body.

Express.js facilitates you to handle GET and POST requests using the instance of express

**API(Application programming Interface)**

Symple example of api:

const express=require("express");

const App=express()  //creates express application

App.get("/people",function(req,res){

    res.send("Hello people");

});

App.listen(3000);

**One more example of Fetching data in JSON(javascript object notation) format:**

const express=require("express");

const App=express()  //creates express application

App.get("/people",function(req,res){

    res.json({people:[{name:"Sudhakar"}]});

});

App.listen(3000);

Get method facilitates you to send only limited amount of data because data is sent in the header. It is not secure because data is visible in URL bar.

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

Web frameworks provide resources such as HTML pages, scripts, images, etc. at different routes.

You define routing using methods of the Express app object that correspond to HTTP methods; for example, app.get() to handle GET requests and app.post to handle POST requests.

var express = require('express')

var app = express()

// respond with "hello world" when a GET request is made to the homepage

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

res.send('hello world')

})

app.listen(3000)

**Middleware (Static files in Express)**

express.static is a built-in middleware function in express

express.static(root,[options])

const express=require("express");

const app=express();

app.use('/static',express.static('public'))

app.get("/",(req, res)=>{

    res.sendFile(\_\_dirname + "/index.html");

});

app.get("/users",(req, res)=>{

    res.send("User's page");

});

app.listen(3000,()=>console.log("server started"));

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

      <title>Document</title>

</head>

<body>

    <h1> hello </h1>

    <img src="static/images/flower.jpg" width="300px">

</body>

</html>