# Unit 1 : Introduction of Node.js Ecosystem

## What is Node.js?

* Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine).
* Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications.
* Nodejs can run on any platform like MacOS , windows , Linux.

**Node.js = Runtime Environment + JavaScript Library**

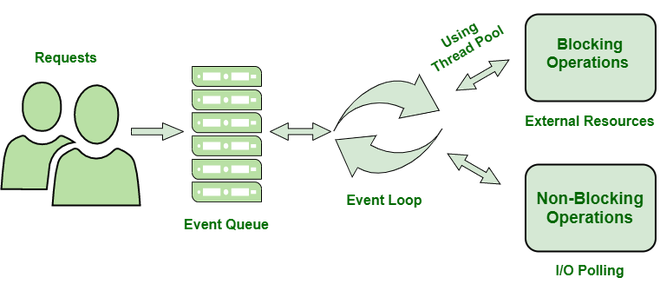
## Features of Node.js

* **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
* **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.
* **License** − Node.js is released under the MIT license.
* Event driven
* Cashing
* No buffering
* Highly scalable
* Open source

## Node.js Web Application Architecture

* Introduction of node js (write in 2 3 lines)
* **web application consists of the following components :- (also write 2 3 lines for this)**
  + **client**
  + **server**
  + **database**
* **Node.js Server Architecture – single thread events loop design / single threaded**

**(also write 2 3 lines for this)**

* + Asynchronous model
  + Non-blocking of I/O operations
* **Components of the Node.js Architecture**
  + **Request**
  + **Node server**
  + **Events queue - *store the incoming client requests and pass them sequentially to the Event Loop***
  + **Thread pool**
  + **Events loop**
  + **External resources**

## Node.js REPL

* **Read eveal print loop**
* The Node.js or node come bundled with REPL environment. Each part of the REPL environment has a specific work.
* **Read:** It reads user's input; parse the input into JavaScript data-structure and stores in memory.
* **Eval:**It takes and evaluates the data structure.
* **Print:**It prints the result.
* **Loop:** It loops the above command until user press ctrl-c twice.

In short we can easily done simply operations with REPL !!

## Prototypal Inheritance using \_\_proto\_\_ in JavaScript

Every object with its methods and properties contains an internal and hidden property known as **[[Prototype]]**. The Prototypal Inheritance is a feature in JavaScript used to add methods and properties in objects. It is a method by which an object can inherit the properties and methods of another object. Traditionally, in order to get and set the [[Prototype]] of an object, we use Object.getPrototypeOf and Object.setPrototypeOf. Nowadays, in modern language, it is being set using **\_\_proto\_\_**.

# Unit 2 : Node.js

## NPM and Modules

* A package contains all the files needed for a module and modules are the JavaScript libraries that can be included in Node project according to the requirement of the project.
* NPM is a package manager for Node.js packages, or modules
* Npm 🡪 dependency installation + install different types of packages

i.e npm install [package name]

- var [package name ] = require(‘package name’)

var express = require('express');

$ npm install express

* For install any module simple use npm install and also for uninstall , update , list ,search use cmd respective.
* NPM can install all the dependencies of a project through the [package.json](https://www.geeksforgeeks.org/node-js-package-json/) file.
  + In the [package.json](https://www.geeksforgeeks.org/node-js-package-json/) file, each dependency can specify a range of valid versions using the semantic versioning scheme, allowing developers to auto-update their packages while at the same time avoiding unwanted breaking changes.

## package.json

* package. json is present in the root directory of any Node application/module and is used to define the properties of a package.
* In package.json file our all installed packages will mentioned with their version.

Project information

• Name

• Version

• Dependencies

• Licence

• Main file

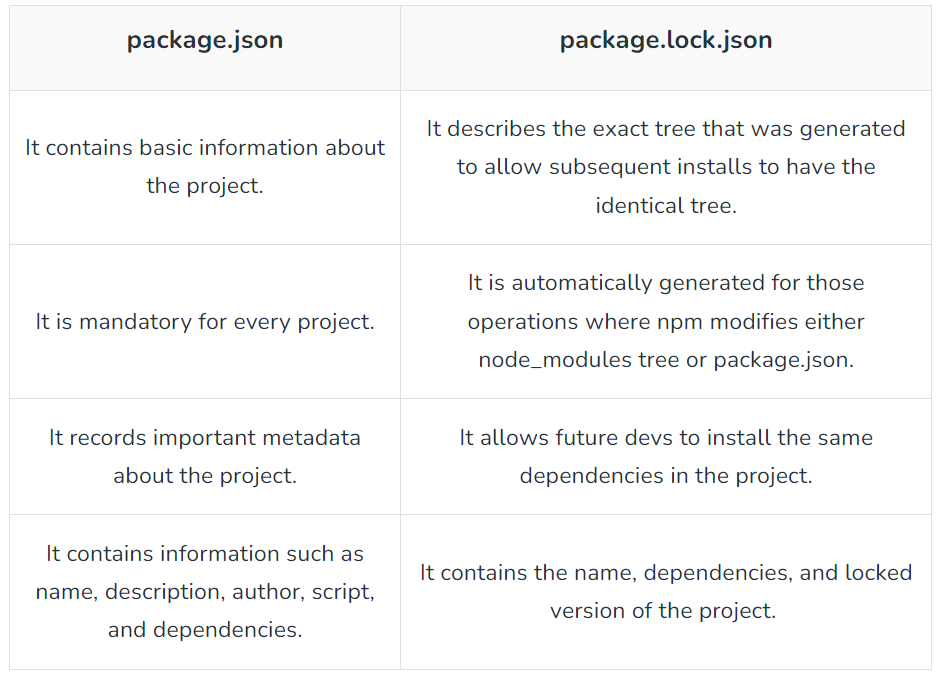
Etc...

* package.json file can create with 2 different ways like
  + npm init then fillup all details accordingly our requirements
  + and another is One can directly write into file with all the required information and can include it in the Node project . (manually create this file)

## package-lock.json

package-lock.json is automatically generated for any operations where npm modifies either the node\_modules tree, or package.json. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

## Difference between package.json and package-lock.json files



\

## Node.js Modules

* <https://www.geeksforgeeks.org/node-js-modules/>
* In Node.js, Modules are the blocks of encapsulated code that communicate with an external application on the basis of their related functionality.
* It can be single file or multiple files collection

**Modules are of three types:**

* Core Modules
* local Modules
* Third-party Modules
* Core modules : inbuilt modules
  + *const module = require('module\_name');*
* Local modules : create by our
  + Create user define logic for some operations and use it into pur nodejs project

*i.e*  exports.myDateTime = function () {

return Date();

};

* Third party modules : external module
  + Npm install mongoose
  + Npm install nodemon

## require(), createServer()

var http = require('http');  
  
//create a server object:  
http.createServer(function (req, res) {  
  res.write('Hello World!'); //write a response to the client  
  res.end(); //end the response  
}).listen(8080); //the server object listens on port 8080

## Node.js Event Loop

* Node.js is a single-threaded event-driven platform that is capable of running non-blocking, asynchronous programming. These functionalities of Node.js make it memory efficient. The event loop allows Node.js to perform non-blocking I/O operations despite the fact that JavaScript is single-threaded. It is done by assigning operations to the operating system whenever and wherever possible.
* **Features of Event Loop**
  + Endless loop , wait for task and execute them then sleep until next task received
  + Execute task only from event queue
  + Allow to use promise and callback

**i.e** console.log("This is the first statement");

setTimeout(function(){

console.log("This is the second statement");

}, 1000);

console.log("This is the third statement");

## Asynchronous Programming

Asynchronous programming in Node.js is a technique that allows you to run multiple tasks at the same time, even if they are waiting for data from other sources. This can be done by using asynchronous functions, which are functions that return a promise. Promises are objects that represent a future value, and they can be used to wait for the results of asynchronous operations.

To write asynchronous code in Node.js, you can use the async keyword before the function name. This will make the function return a promise, and you can use the await keyword to wait for the promise to resolve.

## Callback Functions

A callback is a function that is called when a task is completed, thus helping in preventing any kind of blocking and a callback function allows other code to run in the meantime. Callback is called when a task gets completed and is the asynchronous equivalent of a function. Using the Callback concept, Node.js can process a large number of requests without waiting for any function to return the result which makes Node.js highly scalable.

## Exception Handling

1. **Try-catch Block**

Exception handling in Node.js is the process of dealing with errors that occur during the execution of your code. Errors can happen for a variety of reasons, such as typos in your code, unexpected input from users, or problems with the underlying operating system.

To handle errors in Node.js, you can use the try...catch block. The try block contains the code that you want to execute, and the catch block contains the code that you want to execute if an error occurs.

try {

const fs = require('fs');

const data = fs.readFileSync('myfile.txt');

console.log(data);

} catch (err) {

console.error(err);

}

1. **Throwing Custom Errors(in try catch element)**
2. **Handling Asynchronous Code**
   1. Callbacks

i.e fs.readFile('file.txt', (err, data) => {

if (err) {

console.error(err.message);

} else {

// Handle the data

}

});

* 1. Promises*(.then and .catch function !!)*

i.e somePromiseFunction()

.then(result => {

// Handle the result

})

.catch(error => {

console.error(error.message);

});

1. **Third-party Error Handling Libraries**

## What is REST API in Node.js ?

REST stands for REpresentational State Transfer . A REST API is an application programming interface that adheres to the constraints of REST architectural style and enables interaction with RESTful web services. Interconnected networks make up the web. A web service is a set of open protocols and standards used for exchanging data between client-server applications. Web services that follow the REST architecture are known as RESTful web services.

HTTP methods

Following four HTTP methods are commonly used in REST based architecture.

* **GET** − This is used to provide a read only access to a resource.
* **PUT** − This is used to create a new resource.
* **DELETE** − This is used to remove a resource.
* **POST** − This is used to update a existing resource or create a new resource.

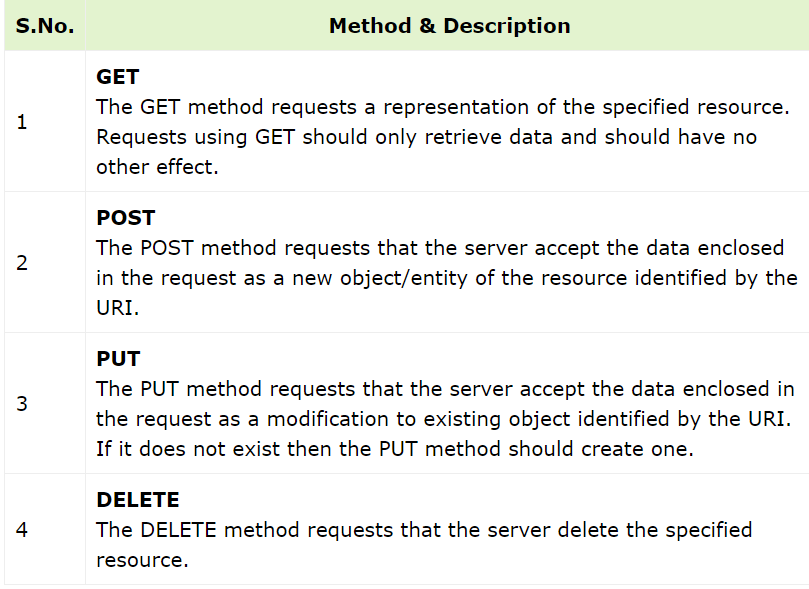
## Mongodb Database(NoSQL Database)

# Unit 3 :ExpressJS

## Routing

Routing in Node.js typically refers to the process of mapping incoming HTTP requests to specific handlers or functions that will process those requests. Node.js does not provide a built-in routing system like some web frameworks, but you can implement routing for your Node.js applications using various libraries and techniques.

## ExpressJS - HTTP Methods



## ExpressJS - URL Building

var express = require('express');

var app = express();

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

res.send('The id you specified is ' + req.params.id);

});

app.listen(3000);

## ExpressJS – Middleware

Middleware functions are functions that have access to the **request object (req)**, the **response object (res)**, and the next middleware function in the application’s request-response cycle. These functions are used to modify **req** and **res** objects for tasks like parsing request bodies, adding response headers, etc.

i.e

var express = require('express');

var app = express();

//Middleware function to log request protocol

app.use('/things', function(req, res, next){

console.log("A request for things received at " + Date.now());

next();

});

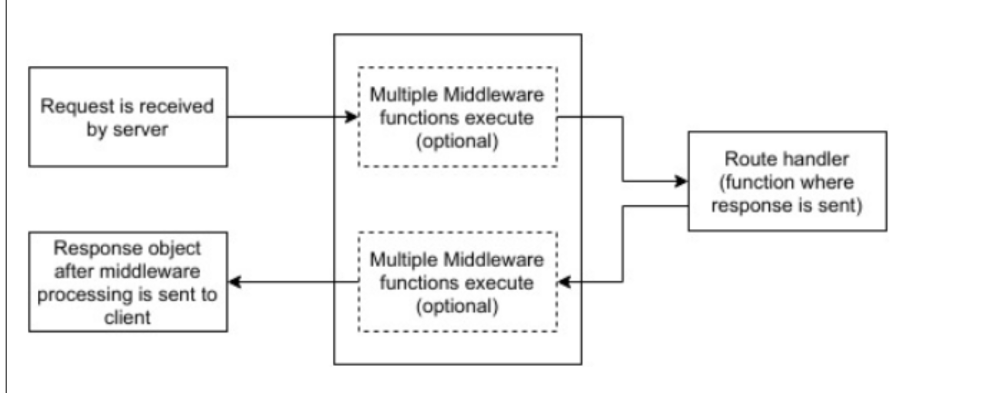
// Route handler that sends the response

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

res.send('Things');

});

app.listen(3000);



## Form data

**In Node.js, you can handle form data from HTTP requests using libraries like express and body-parser.**

const express = require('express');

const bodyParser = require('body-parser');

const app = express();

// Use body-parser middleware to parse form data

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

// Define a route to handle a POST request with form data

app.post('/submit-form', (req, res) => {

const formData = req.body;

// Process the form data

console.log(formData);

res.send('Form data received: ' + JSON.stringify(formData));

});

// Start the Express server

const port = 3000;

app.listen(port, () => {

console.log(`Server is running on port ${port}`);

});

Create an HTML form that sends a POST request to the **/submit-form** route. You can create an HTML file in your project directory, for example, **form.html**:

## ExpressJS – Cookies

we will require the cookie-parser. cookie-parser is a middleware which parses cookies attached to the client request object.

i.e

var cookieParser = require('cookie-parser');

app.use(cookieParser());

var express = require('express');

var app = express();

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

res.cookie('name', 'express').send('cookie set'); //Sets name = express

});

app.listen(3000);

To check if your cookie is set or not, just go to your browser, fire up the console, and enter −

console.log(document.cookie);

You will get the output like (you may have more cookies set maybe due to extensions in your browser) −

"name = express"

////// short form from ppt check it

Install package

npm install cookie-parser

Import cookie-parser into your app.

var express = require('express');

var cookieParser = require('cookie-parser');

var app = express();

app.use(cookieParser());

Define a route:

Cookie-parser parses Cookie header and populate req.cookies with an

object keyed by the cookie names. Browser sends back that cookie to the

server, every time when it requests that website.

Define a new route in your express app like set a new cookie:

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

res.cookie('cookie\_name' , 'cookie\_value').send('Cookie is set');

});

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

console.log("Cookies : ", req.cookies);

});

# Unit 5 Developer tools

Version Control System (VCS)

Following are the types of VCS −

* Local
* Centralized version control system (CVCS).
* Distributed/Decentralized version control system (DVCS).