Node.js

Vraj Suratwala

Department of ICT, VNSGU, Surat

Node.js - Introduction

- What is Node.js ?
- Features of Node.js and Place of Node.js in MERN STACK.
- Advantages if Node.js.
- When to use Node.js and When to not use it.
- Node.js Basic Building Blocks.
- Execution Architecture of Node.
- Blocking vs Non-Blocking.
- Thread VS Async.

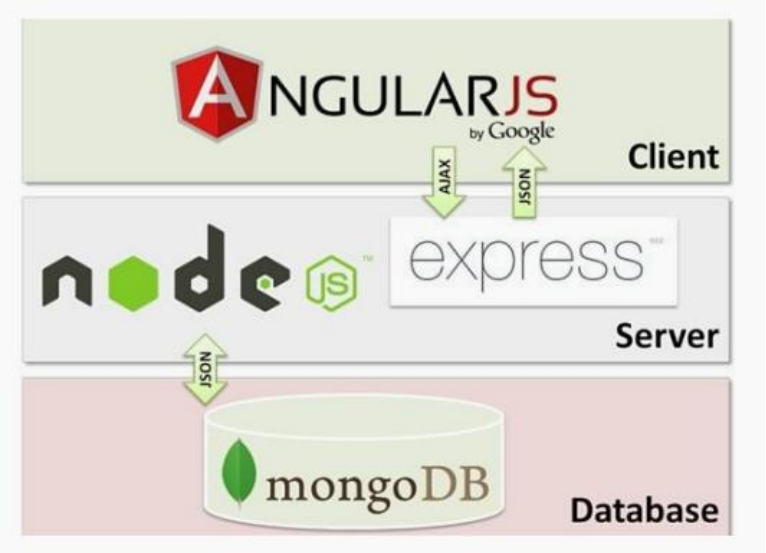
What is Node.js ?

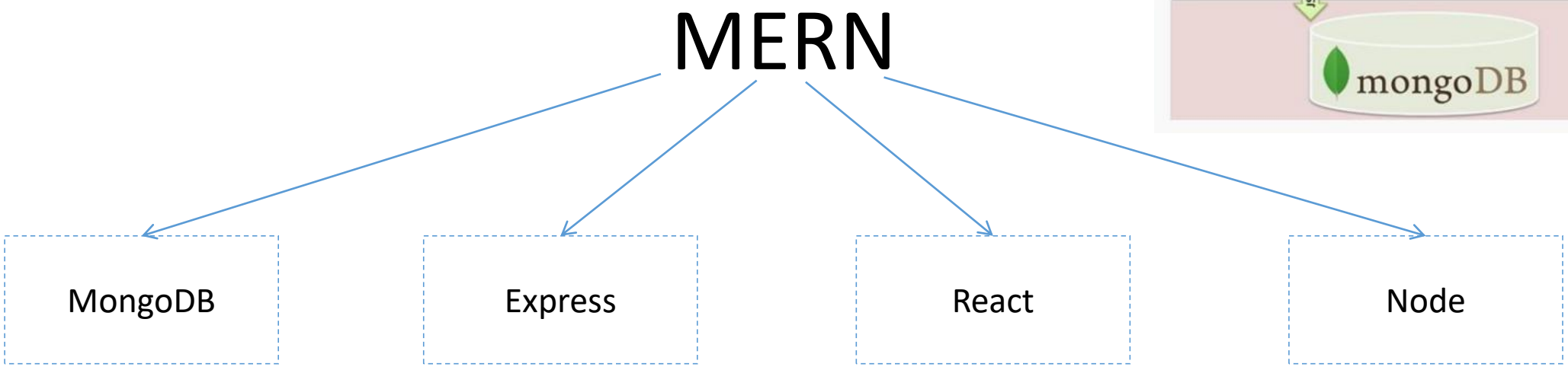
- Node js is Open-Source,Cross Platform javascript Runtime environment which executes out side of a browser.
- Developed in 2009 by Ryan Dahl.
- The Node.js is not using DOM(Document Object Modal) which is used by Browser’s.(in Javascript!)

Features of Node.js

- Highly Scalable
- Extreamly Fast
- I/O is based on Async. way and events
- No buffering
- Open source
- Single Threaded

Node.js in MERN Stack





Advantages of Node.js

- Node.js is an Open-source.
- Uses Javascript to work with server-side functionality.
- Lightweight Framework and it is able to include the other module as per application’s need!
- it is faster because it is using async. way to execute the code and any task.
- also it is a cross platform framework so it can run on any operating system or platform.

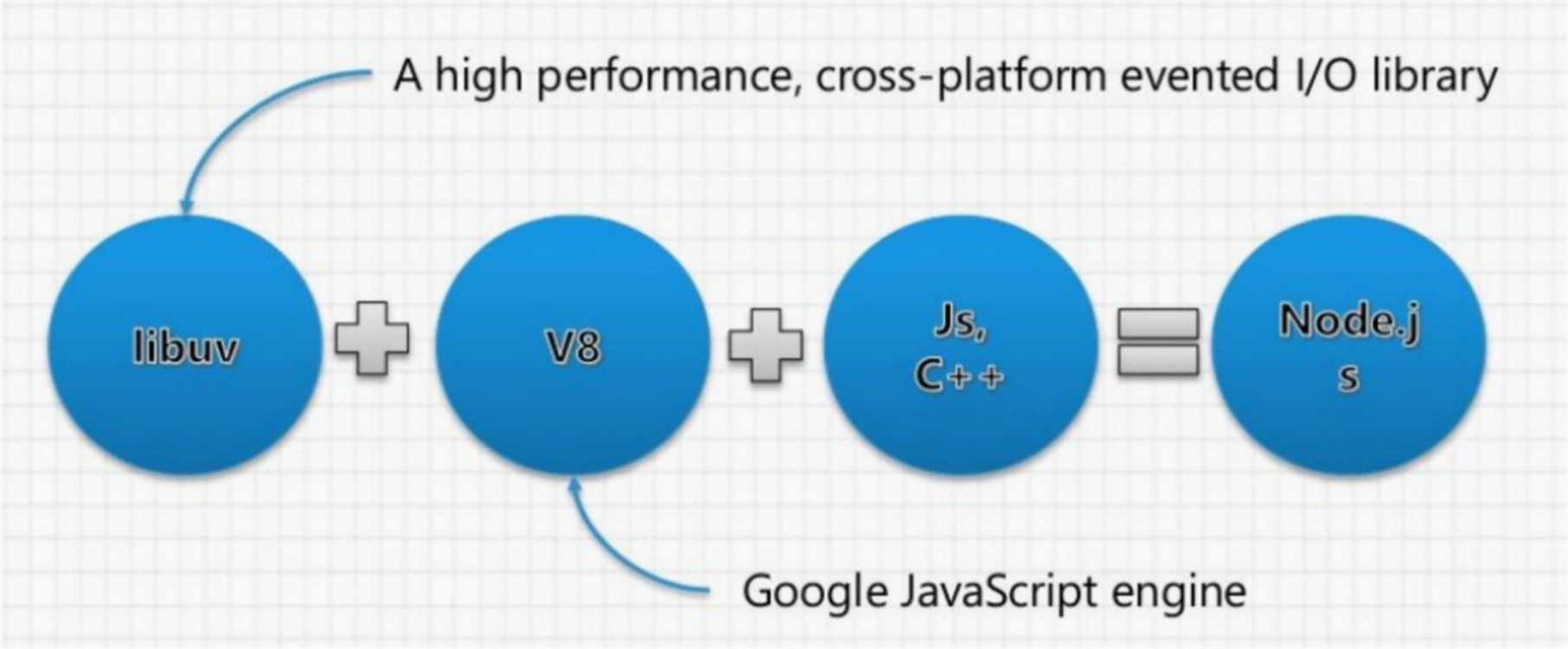
When to use Node.js ?

- Real time application
- Communication Hubs
- Web application
- Web sockets
- Proxy Server
- Streaming Server etc.

When to not use Node.js?

- in Real-time Application that require high pricision timing.
  - because, JavaScript timers are not always precise, and Node.js is subject to event loop delays under high load.
- CPU-Intensive Applications.
- Application require the strong Type Safety
- in Complex Multithreading.
- Low level system Programming.
  - such as kernal development, device drivers, Embedded System that oftenly written in c,c++,assembly or rust etc...

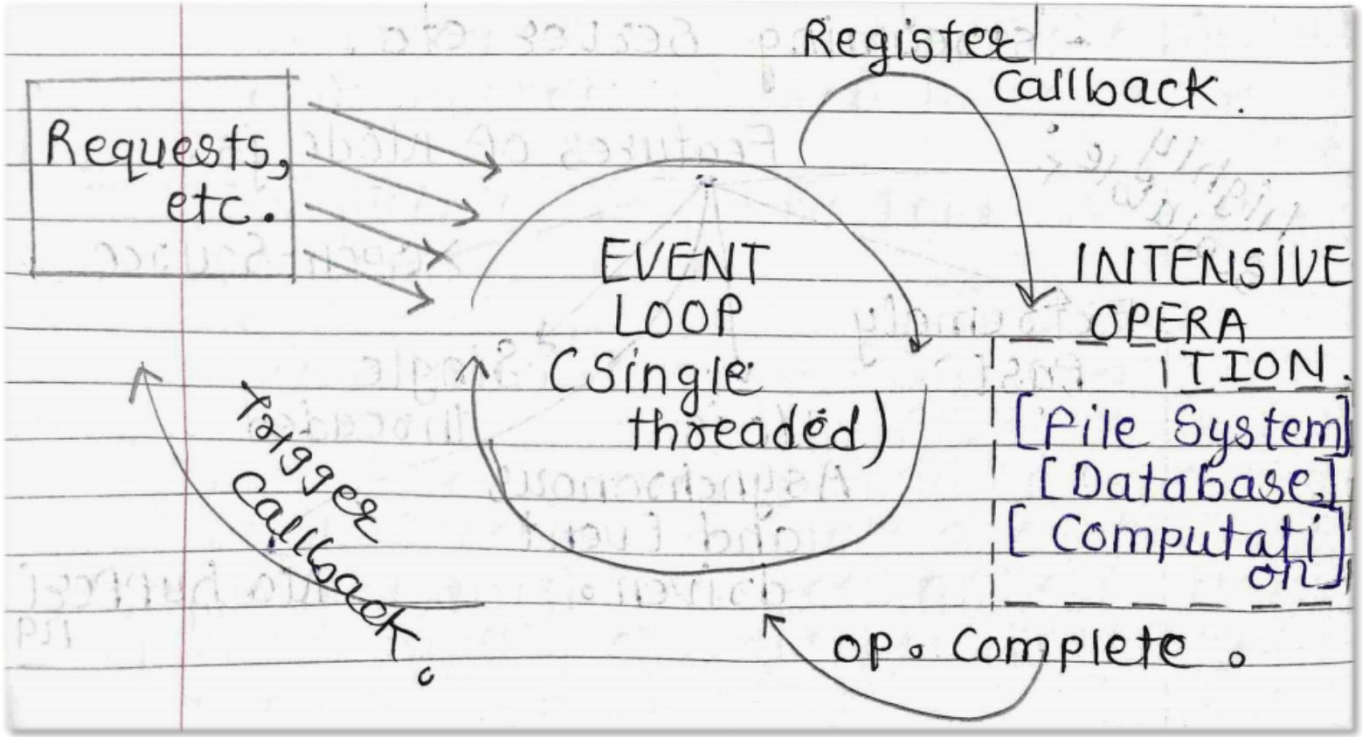
Node.js Blocks



Execution Architecture of Node.js

- It is devided into four parts.

- 1. Client Request Phase
- 2. Event Loop Phase
- 3. Request Processing Phase
- 4. Response Phase



Execution Architecture of Node.js

- 1. Client Request Phase
  - Client send the request to node.js server.
  - each request will be added to an event queue.

- 2. Event Loop Phase
  - it is continously checking the event queue.
  - it will pick the request one by one from event queue.

Execution Architecture of Node.js

- 3. Request Processing Phase
  - simple non-blocking task will be handled by main thread.
  - complex and blocking task will offloaded into thread pool.

- 4. Response Phase
  - when blocking task complete, their callbacks are placed in the callback queue.
  - an event loop process callbacks and send response.

  - So, This is an Execution Architecture of Node.js.

Blocking vs Non-Blocking in Node.js

Characteristic	Blocking	Non-Blocking
Execution Flow	Synchronous	Asynchronous
Performance	Less efficient	More efficient
Code Complexity	Simpler	More complex
Error Handling	Try/catch	Callback errors/Promises
Use Case	Simple scripts	Scalable applications

Thread VS Async. Notes

Threads.	Asynchronous Even-driven.
- locks an app and request with listener-works & threads.	- only one thread, which repeatedly fetches an event.
- using incoming request model.	- using queue and then processes it.
- using Context Switching.	- No Context Switching.
- uses multithreading.	- uses asysc I/O.
- multithred Server might block a req. which might involves multiple events.	- Manually saves state and then goes on to process the next event!

Example :

```
1   import os from 'os';
2   import fs from 'fs';
3
4   console.log(os.cpus().length);
5
6   console.log(1);
7   console.log(2);
8   // Blocking
9   console.log(fs.readFileSync("./hello.txt","utf-8"));
10
11  // UnBlocking
12  console.log(
13      fs.readFile("./hello.txt", "utf-8", (err,data)=>{
14          if(err)
15          {
16              console.log(err);
17          }else{
18              console.log(data);
19          }
20      })
21  );
22
```

Node.js - Basic

- Concepts - callback(), REPL Terminal, Data Types
- Types of Error and Debugger
- Modules in Node.js
- Global Objects in Node.js
- console() methods
- EJS vs CJS
- json.package and json-lock.json
- Packages in Node.js
- npm - node package manager and it’s Commands.

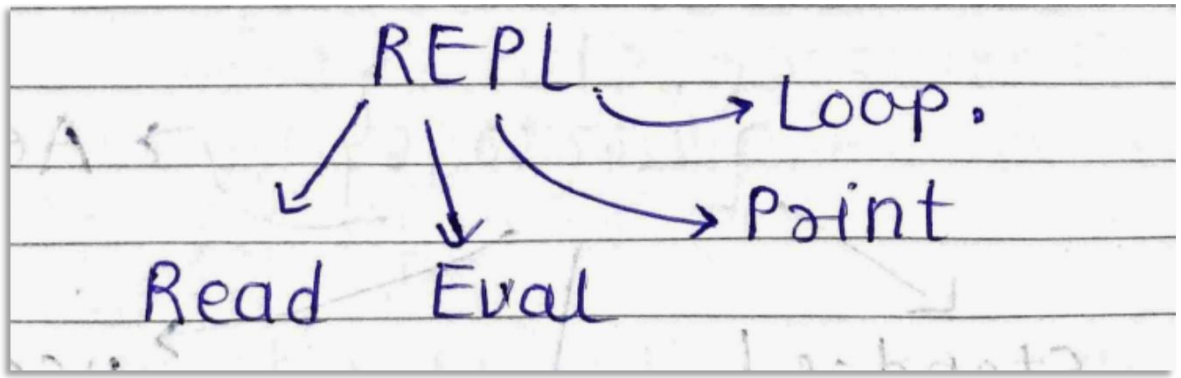
Callback()

- callback is an async. function that called at the end of the task.
- Node makes a heavy use of callbacks.
- all the api written in the callbacks.
- example of callbacks.

```
// UnBlocking
console.log(
    fs.readFile("./hello.txt", "utf-8", (err,data)=>{
        if(err)
        {
            console.log(err);
        }else{
            console.log(data);
        }
    })
);
```

REPL Terminal

- Node comes with bundled of REPL Environment.
- Basically, It is providing the the terminal like linux shell in which user can entered the command and system will respond the response.
- as we can seen in the daigram.
- REPL refers to Read, Eval, Print, Loop!



REPL Terminal

- 1. READ : the system will reads users input and will be parse the data in js format and stores in the memory.
- 2. EVAL : it will takes and evalute the Data Structure.
- 3. PRINT : Prints all the results.
- 4. LOOP : Loop the above command until the user presses CTRL + C.

REPL Terminal

- The REPL feature of Node.js is very useful in expermenting in node.js codes and debug javascript codes.

- in order to run the REPL Terminal.
  - $node

Node.js Data Type

- Node.js includes primitive Data types.

- String
- Number
- Boolean
- Undefined
- NULL
- Regexp
- Object

Error is Node.js

- 1. Standard Error
- 2. System Error
- 3. User-Specified Error
- 4. Assertion Error

Various Debugger

- 1. Core Node.js Debugger
- 2. Node inspector (npm i -g node-inspector)
- 3. Built-in Debugger in IDE’s.

Modules in Node.js

- Modules refers to Javascript Library.
- the set of functions that we want to include in our application.
- Two Types of Modules.
  - 1. Built-in Modules
  - 2. User-Defined Modules

1. Built-in Modules

- Node.js is having the set of built-in modules which we do not need to install externaly.
- in CommonJS , to use the built-in modules, we need to use the require() function.

- syntax :
- var/const variable_name = require(‘module_name’);

1. Built-in Modules

- in ESM we have to use ‘import’ keyword.

- Syntax :
  - import variable_name from ‘module_name’;
- Example :
  - import fs from ‘fs’;

2. User-Defined Modules / External Modules

- it can also refered as a user-defined modules.
- for that we have to create,import and use this module as shown in below example.

math.js

```
function add (a, b) {
    return a+b;
}
module.exports =add;
```

main.js

```
var math = require ('./math.js');
Console.log (math.add (10,20));
```

O/P.

```
30.
```

Node.js Global Objects

- Node.js Global Objects are global in nature and they are available in all modules.
- we do not need to include it externally.
- we can include it directly.
- this objects are function,string and object etc.
- in node.js there is various kind of global objects.

Global Objects

- 1. __filename : this will return the file name which is going to be execute now.
- Example : console.log("File_Name : " + __filename);
- 2. __dirname : this will return current directory path in which you are working.
- Example : console.log("Directory Name : " + __dirname);

Global Objects

- 3. setTimeour(cb,ms) : global function is used to run callback cb after at least ms milliseconds.

- Example : function printhello()

```
{
   console.log("Hello User!");
}

setTimeout(printhello,5000);
```

Global Objects

- 4. clearTimeout(t) global function is used to stop a timer that was

previously created with setTimeout().

Example : var time = setTimeout(printhello,5000);

clearImmediate(time);

5. setInterval(cb, ms) global function is used to run callback cb

repeatedly after at least ms milliseconds

Example : function printhello()

```
{
   console.log("Hello User!");
}
setInterval(printhello,5000);
```

Global Objects

- 6. console
- 7. buffer
- 8. process

Note: a process object is a global object that gives information about and controls the node.js process!

- Reference : https://www.geeksforgeeks.org/node-js/node-js-global-objects/

console () - Methods

- console.log()
- console.error()
- console.warn()
- console.info()
- console.dir(,[,objects])
- console.time(lable)
- console.timeEnd(lable)

ESM vs CJS

- in Node.js EJS and CJS refers totwo different type of module system.

- 1. CJS : Common Java Script
  - CJS is refers to Common Java script Module which is Original System used by Node.js.
  - to import an module or library it uses import() and to export the module and library it uses module.exports().

  - Syntax: (Import)
  - var/const variable_name = require(‘module/library_name’);

  - Example :
  - const fs = require(‘fs’);

ESM vs CJS

- 2. ESM : ECMAScript Module.
- it is a mordern way for node.js

Syntax : import variable_name from ‘module_name’;

export function_name

export default function_name

ESM : Example

```
1   import fs from 'fs';
2
    1 reference
3   function reading()
4   {
5   fs.readFile('./hello.txt', 'utf-8', (err,data)=>{
6       if(err)
7       {
8           console.log(err);
9       }else{
10          console.log("Data is Here");
11          console.log(data);
12      }
13  });
14  }
    0 references
15  export default  reading();
```

ESM vs CJS : Summary

Feature	CJS	ESM
full-form	Common Java Script	ECMAScript Module
File-Extension	.js	.mjs or .js (with type = “module”) in package.json file
way	Traditional	Modern
Syntax	require() module.exports	import and export keywords
loading of data	Syn.	Async.
Usage	Traditional Node App	Mordern JS and Node apps

Package.json and Package-lock.json

- in any Node.js Project, these two file are manageing the dependencie and help ensure consistent project behaviour iiin different environment.

- the package.json is main configuration file in node.
- which contains..
  - name
  - versions
  - main
  - scripts
  - author and license
  - dependencies
  - devDepencies etc...

Package-lock.json

- this file will be autometically genarate when we create install a package using ‘npm install’.

- it will loack the versions of installed dependencies.
- Use Cases :
  - Lock Dependecies : ensures that exact version of the packges and sub-packages.
  - Faster Install : Speeds up npm install by avoiding version resolution.
  - Consistency and Security.

Package.json VS Package-lock.json

Feature	Package.json	Package-lock.json
Required by npm	YES	Auto-Generated
Describe-Dependencies	YES - Sementic Version (Example : 1.0.1)	YES - Exact Version
Human-Editable	YES	NO
Ensure Consistency	NO	YES

Packages in Node.js

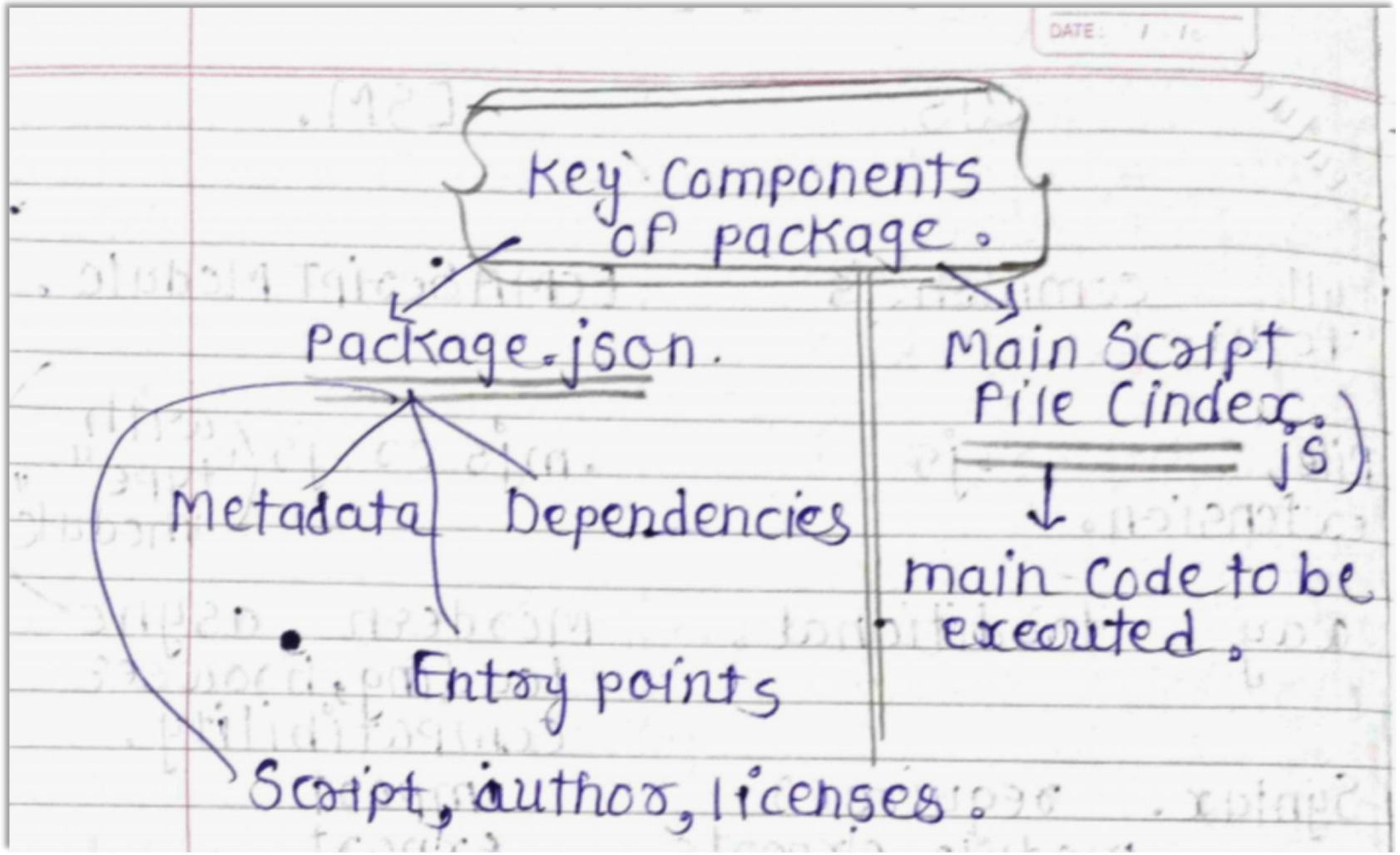
- a package in Node.js is a directory or folder thatcontains module or a collection of a modules along with some package.

- Usage : It is used to organize and manage the code reusability.
- Other Definition : Package is a reuseable piace of code that can be included in your project to be extend functionality without writing everything from base.

Packages in Node.js

- We can also define as a module which contain the reusable code.
- includes the package.json which includes the metadata about the component of Packages.
- can be installed locally and globally using commands.
  - Local : Example - npm i nodemon
  - Global : Example - npm i nodemon -g
    - Note : ‘-g’ is used to install a package globally.

Key Components of Packages



Types of Packges

- 1. Core Packages
- 2. Third-Party Packages
- 3. Custom/Local Packages

1. Core Packages

- The core packages in the node are already built-in packages orr modules that you do not need to install externaly from anywhere.

- it comes when you install the node using npm i.
- Example :
  - const fs = require(‘fs’);

2. Third-Party Packages

- This Thired Party Modules or any Packges must be installed manually if you want to use it.
- for an instance it we want to use it so we can also install from the npm.js.
- Example : Express
- Mongoose
- Nodemon
- dotenv etc..

- Example : npm i express

3. Custom or Local Packages

- it can create by you or your team along with package.json and exported a function.
- installing a package : npm i express

How to Create a Package in Node.js ?

- Step 1 : Create a folder.
- Step 2 : Intialize with package.json file.
  - using ‘npm init’ command.

- Step 3 : The Code shourld be based on the ESM or CJS as given in the example above.

npm - Node Package Manager

- npm stands for Node Package Manager which default package manager for Node.js.
- allows you/us to install,manage and share JS Packages. (libraries/Modules).
- Comes autometically when you install a Node.js.

Why we should use npm?

- Purpose :
- Install packages - npm i express
- Manages Project Dependencies - Package.json and Package-lock.json
- Run Scripts - npm start and npm run dev
- Share Your own Packages - npm publish

Command npm commands with Example :

- npm init
  - intialize a project and Creates new package.json.
  - sets project name,version and etc...
  - Example : npm init

- npm init -y
  - create default package.json without prompts.
  - Auto-Genarate Meta-Data.

Command npm commands with Example :

- npm i <package> : installing a specific package in your application.
- npm i <package> --save-dev : install a package and add it into dependencies and Devdependencies too.

- Example : npm i nodemon --save-dev
- Other Commands :
- npm start
- npm publish
- npm run <script> etc...

Thank You