Unit-6

→ Node-JS Introduction - what can Node.js do - Features - Projects - synchronous & asynchronous.

Node.JS → opensource Server environment
↓ can able to run JS
↓
Run-time environment for JS
→ Code can be executed in the computer process
→ example one example in settings - dev.tools - alert("hello");
→ In GC (Googlechrome) | IE (chakra) | MF Spidermonkey
V8 (Engine)
↓ (own javascript engine → Provide runtime JS Environment for a JS code)
→ Irrespective of browser if we install node-JS it will provide that environment → serverside Applications → we can develop fully funct applications

→ It is cross-Platform (run in windows, linux, unix)
→ Very fast to execute

→ synchronous programming
→ execution line by line
↓
line execution depends on Previous line
→ Blockage

≡

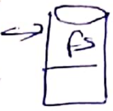
Asynchronous Progm
→ Execute randomly
→ can execute n no-of lines
→ no blockage

→ server side Programming (PhP/ASP)
↓
synchronous
→ node-JS is also server side programming
↳ Asynchronous. → no blockage of process

→ In PHP/ ASP (synchronous)

1. Read

2. wait →



3. ← Response to client

4. → Ready to handle 2nd req.

In PHP

```
→ Echo "hai"; ← 1st
{ $J = f (a.txt); ← 2nd
  echo "end"; ← 3rd.
```

1st Req

read →

2nd req →

(non-blocking)

→ execute in the background

→ no waiting process

→ less time

→ Asynchronous Programming - non-blocking

→ using single thread

(Every request is shared among themselves

Pgm

```
→ console.log("hai");
→ function (".text") { also call back
                        function(){
               });
→ console.log("end");
```

Every line is independent of each other

Node.js → Runtime environment for JS

→ create dynamic webpages

→ collect form data

→ we can Perform add, delete, modify, of DB.

→ Built Backend services + Serverside applications + n/w applications.

→ create, open, read, write, delete in the file system,

Features

→ Asynchronous Programming → never waits for Previous sends

→ Very fast → Googlechrome V8 engine

→ single thread (large no of requests)

→ no Buffer

→ Projects → shopping applications (etc).

chat application, live update, social N/w

→ single Page applications → gmail, github.

→ Except CPU related works, it can do everything

Example explanation

After the example, create button.

b1
b2
b3
b4
toggler

② Node.js Installation - REPL Terminal - first Application Execution -

Components:

-> Install node.js -> runtime environment

->download node js (process explain)

-> later alert ("hai")/ In console -> by pressing f12 key.

-> 5+2=7

-> console.log("javascript");
   ->javascript.

In cmd prompt check whether node.js installed properly or not.

```
> node --version. (you will get version)
> npm --version
> node -v.
> npm -v
```

>create a folder nodejs all the programs must be saved there.

> console.log("hai");
   ->Error

->node environment (REPL - Terminal) (By using node command)

```
>e:
>cd nodejs
e:\nodejs>node
>2+3
5
```

```
> console.log ("node javascript");
> 1 + (3 -2)
> 2
> Var Y=10.
> Var X =5.
```

→ ctrl+c → To come out of loop

```
→ do{
    j++;
    console.log ("j:"+j);
  }while(j<5);
```

→ This terminal can be used to execute any kind of JS code.

→ Read – Evalu– Print – Loop

- Read ↓ user i/p + store in corresponding memory
- Evalu ↓ evaluate given data
- Print ↓ Print in the terminal

} This process is going to loop n no. of times.

→ ctrl+d → to come out of loop.
(or)
(Ctrl+c 2 times) (or) (.exit)

→ tab → all commands
→ .help

Node.js App – 3 components

1) Importing req. modules
2) Create server → handle all the req
3) Read request & return response
   ↓
   from browser/console.

→ open notepad

write → console.log("welcome to node JS execution from file in REPL Terminal");

Save it with jv.js → In nodejs folder

→ now open Command prompt

→ node jv.js (can do without extension also

→ o/p

odules in Node.js _ local Module _ core module - Local Module creates

Module → group of code to satisfy a particular functionality
↳ Reuse the code

→ In nodejs we call reusable code as modules
↳ simply a JS library
↓
collection of various functionalities

→ ① Import the module

In nodejs we have 2 types of models

1. local Mode

2) core modules. (Predefined modules)

1) http (web models – (classes, methods, events & http server)

2) url — Parse the url.

3) query string – to deal with query string

4) Path – filePaths ('open [illegible].eng.rgukt and ? mark)

5) fs (file system) – To work with files in filesystem

6) util Module
↳ Purpose is (taking i/p from console)

→ eg rguktsklm.

→ once we develop local model

{ ≡ ≡ at last we need to write module.exports = ?
→ object (Act as exposed as modules)
↑
represent current module (current group of lines of code)

→ How we are using core module.

→ We have to import local modules with the help of require function.

→ rule → require ('./_____')
↓
modulename.

(or) require ('http')
↓
for core module.

Execution procen

1) notepad →

```
1. Const college={name:"rgukt sklm", Year:2016, strength:40
2. module.exports = college;
```

]→ jv.js ⇓ ① file

For example we have created

→ jv.js

Rewsable Code.

→ some one wants to use this file → then he need to create his own file "main.js"

→ In order to reuse it he needs to use require('./jv ') ⇓ (filename is modulname)

② main.js

```
→ Var j=require('./jv')
Console.log("name of the college is:"+j.name)
```

→ Go to cmd.

```
E:\> cd nodejs
>node main.js
```

– ② file

→ gives o/p

→ you can also modify above file.(include year & strength).

(→ we can also take like

jd.js

v~~ishnu.js~~ ←

③ file

```
class college{
    constructor(name, Year, strength)
    {
        this.name=name;
        this.Year=Year;
        this.strength=strength;
    }
    → wishes(){ console.log("name of college:", ~~this.~~ name, this.name);
}
module.exports=college)
```

→ edit main.js → file like this

```
Var college=require('./jk');
Const p=new College('rgukt nuz',2008,6000)
p.wishes();
```

→ now execute node main.

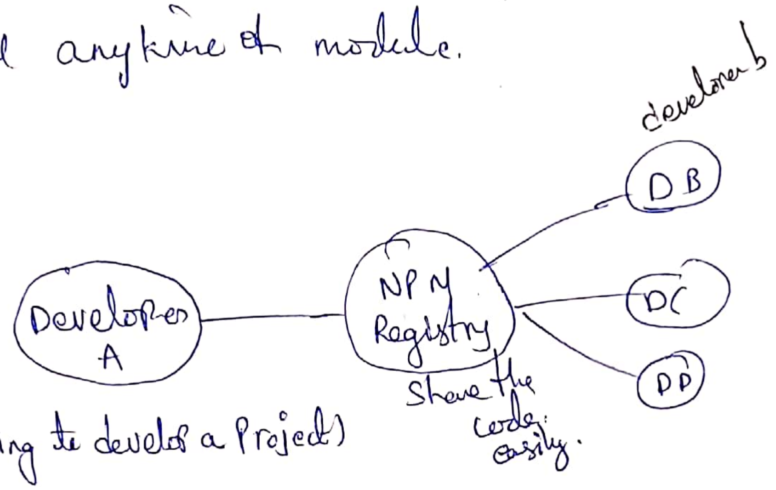
NPM (node.js) (node Packet manager)

→ If multiple People try to work on same Project and every Person having own Predefined moduls. and combining all this modules. will take more time

→ Install any kind of module.

cmd
(npm help)

Eg:



(Trying to develop a Project)

→ * npm --Version.

(or)

→ * npm -v.

To check for list of help commands

> npm heep

→ we can installe any ~~command~~ kind of module by using.

* → npm install -h

*
→ npm init

then Introduce Package name → About to write to, E:\nodejs\Package.json

Package name: (nodejs) vishnu

description: vishnu nodejs program

entry Point: (main.js). → (Then In nodejs folder you observe Package.json file)

→ yes.

Next →

npm init, (create Package.json → manage dependency (muetiple modules).

→ And (~~If~~ you want, after delete Package json file in nodejs folder) then in the command Prompt execute like

→ npm init --yes.

→ Package: vishnu
Version: 1.0.1.]→ enter details like this

→ Package.json → define all the things related to a Project.

① → npm install <module-name>

Eg: npm install moment

Moment → Parse, validate, manipulate, display date & time in JS.

② npm install lodash

→ To get the o/p in consised manner

③ npm install -h

→ If we want to see the consuming space observe the following thing.

→ Del- nodejs Package.json → folder

→ But in Package.json it contains all the things

→ But again by doing > npm install

→ If A person develop lodash & so on. In Package json everything will be installed.

If we want to uninstall then

A > npm uninstall moment

> npm uninstall lodash

Eg: npm install moment --save-dev →(development Package)

> npm install lodash@3.3.0. → 3.3.0 → Patched version
↓ major version ↓ minor version

→ can also install framework

We also have npm install update command.

→npm update lodash.

we have

npm init
install
uninstall
update

→shorthand →

① npm init --yes → npm init -y
② npm install lodash → npm i lodash

npm --version

Advantage → can install & also uninstall
→multiple people can be able to work.

VS NPN scripts - Task Runners - Node.JS:

Module
local core
+ +
Example example
→NPM
(init, install, etc.)

→explain the script in Package.json.
→ To automate the building of Project,
NPM script → JSON key value pairs | & automate repetitive tasks.
↓
Package.json → commands + scripts

```
npm run <scripts>
```

⇒ we write them in JSON key value Pairs
→ Available in Package.json file & hold various Commands + scripts

→main.js

```
console.log("hai main");
```

→ npm run test.
→ make changes in Package.json file

```
"scripts": { "start": "node main.js" } → npm run start
```

,"say_hello": "echo 'welcome'"

> npm ~~say-hello~~ run-script . say-hello. } → Automatically building of project

> npm run --silent say-hello.

→ automatic repetitive tasks.

```
> "scripts": {
   "start": "node main.js",
   "say-hello": "npm run start && echo 'Hello world!'"
      }
```

> npm run say-hello.

Task runners: (to automate the repetative work

↳ collection of tasks / Repeatitiveness / minification of code.

CSS SASS

section{ h:
 w:
}

{ In order to convert from css to sass or from sass to css, we need task runners

section .class-one{ h: w: }

① Grunt.] uses → nodejs as a platform | Its purpose is to automate anything with minimal effort.

② Gulp

> npm install golp -g.

> npm install grunt

Core module Except web module→(http)

→ main.js file

working with url module (purpose to check url)

const url = require('url');

const myurl = new URL("https://www.rguktsklm.ac.in:8080/careers/jv.html?id=535&name='priya'");

console.log("my url is", myurl.href);

console.log("my host is", myurl.host);

Path → myurl.path;

Query string → myurl.search

hostname → myurl.hostname

→ const os = require('os');

> console.log ("my Platform is", ~~my~~ os.platform());

> console.log ("my architecture is:", os.arch());

> home directory → os.homedir();

total memory → os.totalmem();

free memory → os.freemem();

cpu related info → os.cpus();

→ Path

const Path = require('Path');

> console.log ("my base name is :", Path.basename(_filename));

> console.log ("my directory is:", Path.dirname(_filename));

> extension → path.extname (_filename);

> single line (properties) → Path.Parse ("

→ combining of 2 Paths , Path.join (_dirname, 'test', 'hello.html'

→ fs module

```
const fs = require('fs');
 fs.mkdir('/test', {}, function(err){
     if (err) throw err;
     console.log ("folder created");
 }).
```

→ folder will be created in your C drive in corresponding nodejs directry.

(or)

```
const Path = require('Path');
  fs.mkdir (Path.join (_dirname, '/test'), {}, function(err){
    if (err) throw err;
     console.log ("folder created");
  })
```

→ web Module (http).

→ Transfer data over http.

1. require('http');
2. createServer();
3. listen();

port number → 65,535

0 to 1023.

cse.js →

```
Const http = require('http').
http. createServer ((req, res) => {
 res.writeHead(200, {'content-type': 'text/plain'});
 res.write ('welcome to nodejs server');
 res.end();
 }).listen(4231)
```

→ instead of this we can also write

(res.setHeader('content-Type', 'text/html');

(localhost:4231).

Create a html program (first.html).

```
const http =require ('http');
const hostname =' 127.0.0.1' ;
const port =4231 ;
const fs =require ('fs');
 fs.readfile (' first.html' , (err, data) => {
     if (err) throw err;
const server =http. CreateServer ((req, res) => {   →   res.setHeader('content-Type', 'text/html');
                                                         res.write( ~~welcome~~ data ');
                                                         res.end();
     })
server.listen (port, hostname, () => {console.log ("Server started at port
                                                   number:", port)})
```

→ first.html

```
<html> ~~<head>~~
<body style = "background-color: yellow;">
  <h1 style ="color:blue;"> welcome to http Server loaded Page </h1>
  <hr>
   <hr> jv </h5> </body></html>.
```

vs Express framework to respond to http Server

- → Routing tables, Setup middlewares
- → Allowing dynamically rendering of web Pages
- → can install Plugins.
- → easy, Robust & quickway of setting API's
- → more flexible & pluggable

→ no need to use specific structure
→ Always upto developer point

"express" module Impoted to our project

→ npm install express --save.
→ npm i express -save.

→ main.js

```
① Var express = require('express');
```

Go to
```
>cmd npm init
Package: jvexpressproject
Version: 1.0.2
des: jvexpress basic idea
key: jv
auth: ~

>npm i express --save.
```

now go to program

```
② → var app = express();
    app.get('/', function(req, res){
      res.send("welcome to express framew");
    });
    app.listen(3000);
```

```
→ app.get('/vishnu', function(req, res){
    res.send( "vishnu priya");
  })
```

```
(③) use
  app.listen (3000, ()=> {console.log("server started at port 3000")});

  app.get ('/cse', function(req, res){
    res.send ("cse branch");
  })
  app.use (express.static('public'));
```

(create a folder. Public & create a program first.html & save in Public)

(localhost:3000/first.html)

est -full API (Application Programing Interface)

webservice
- SOAP (Simple object Access protocol).
- REST full-API

(covid19.org)

Representational state Transfer

(contains set of rules, developer need to follow while developing an API)

→ REST is an architecture used to create rest full server.

→ Based on http protocol.

Create - POST - create Resource
Read - GET - Retrieve
update = PUT - change/update resource
Delete - Delete Response.

→ Inorder to work will all these we need Postman tool.
↓
Verify all API's.

⇒ Download Postman tool.

→ Create a new file. (rfa.js)

```
const  express =require('express');
const app =express();
const port = 4231; → app.use(express.json());
app.get('/', (req,res) =>{
        res.send("working with restfull default API");
})
app.listen(port, () => console.log("Server starting at port number", port));

let books =[ { id: "101", title: "wt", author: "sebasth" }, ]
app.get('/bookslist', (req,res) =>{
        res.json(books);
}).
```