Marwadi
University

01CE0412-Advanced Web Technology

Unit-3 NodeJS

Prof. Vaibhav K Matalia
Computer Engineering Department

Introduction of NodeJS

- Node is not a language. This is a server environment.
- Node.js can connect with database.
- Code and syntax is very similar to JavaScript. But not exactly the same.
- Node.js free,open-source.
- Node.js use Chrome's V8 engine to execute code.(similar to C,C++ compiler)

Why do we use node?

- Node is mostly used for API. So we can connect the same database with Web App,Mobile App.
- Node is easy to understand who know javascript.
- Node is super fast for APIs.(because it uses V8 engine)
- With Node and JavaScript, you become full stack developer.

History and More:-

❖First release:-May 27,2009

❖Current Version:-20.10.0

❖Written in C,C++,Javascript

Note:-file system, database connectivity code written in C and C++ and Major code is written in Javascript.

Javascript & Node are the Same?

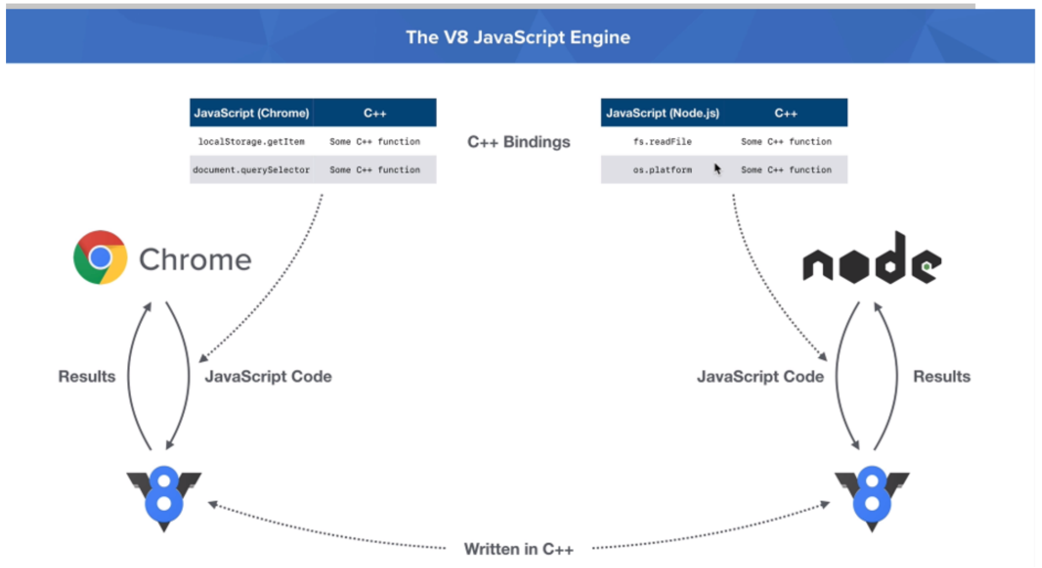
- Javascript and Nodejs code syntax is same, If you know javascript you can easily understand Node. But both are not exactly same.
- We can not connect javascript to Database, Node can connect with database.
- Nodejs run on server side, JavaScript run on the browser.

Client and server:-

- When we access something from internet, two machine(Client, server) which is involve in this communication.
- HTML,CSS,Javascript executes in client machine.Node.js execute on server.
- When we write name of website at that time one request will goes to the server and the nodejs code will be executed in the server.Node.js can be connected to database, the file system can be used.Client will get data through API.
- Node.js uses JavaScript internally.

How node.js use Javascript?

- Chrome uses v8 engine to execute JavaScript code. If you execute javascript code in google chrome, then chrome sent code to V8 engine and v8 engine will give the result to chrome.
- Whoever created Node thought that if they use Chrome’s v8 engine, then they can connect JavaScript to the backend as well.
- v8 engine is built in C++.



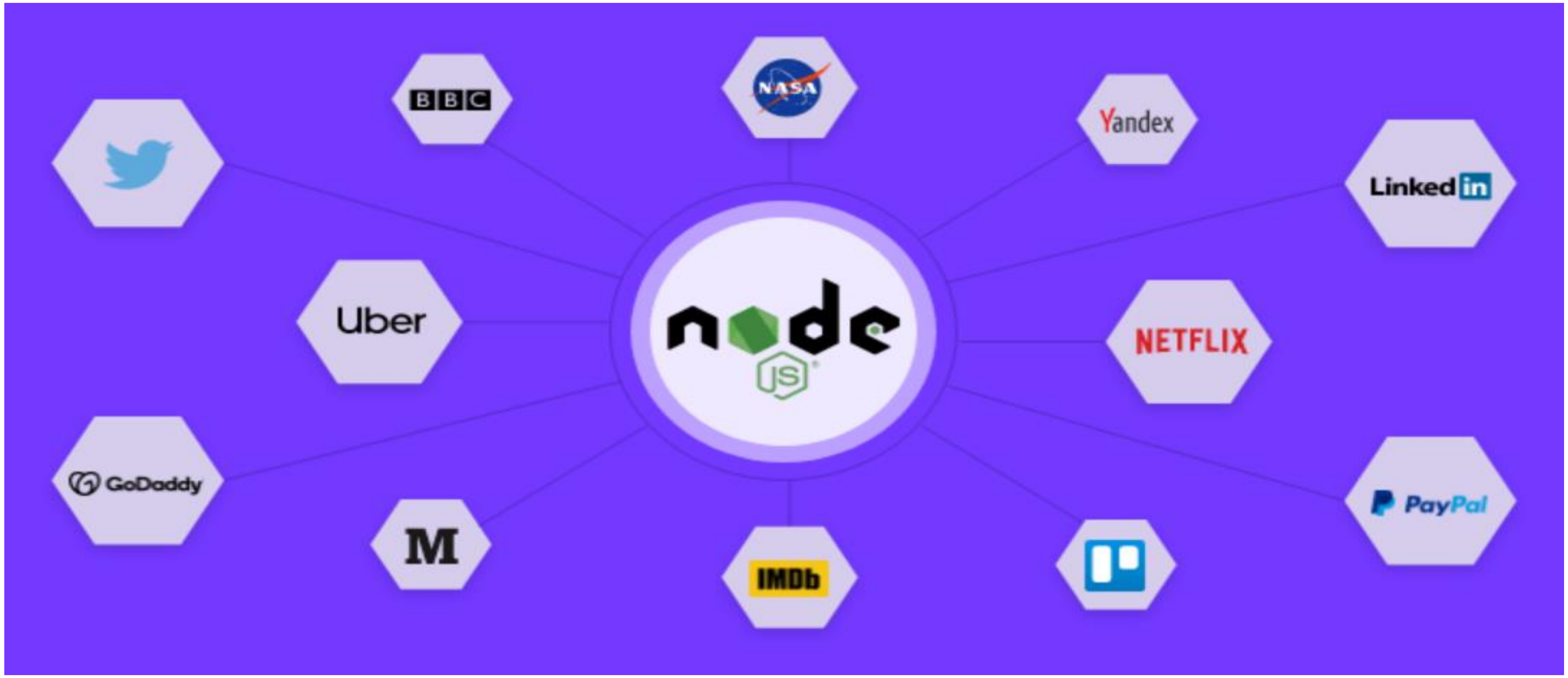
JavaScript (Chrome)	C++
localStorage.getItem	Some C++ function
document.querySelector	Some C++ function

JavaScript (Node.js)	C++
fs.readFile	Some C++ function
os.platform	Some C++ function

What developer make with Node.js?

- Developers make API with Nodejs.So we can connect server with client.
- Node can make API for web, android and IOS etc.(Same API for Web,Android and IOS).
- A website can also be created from node but it rarely happens.
- When request has come from mobile app or web app at that time server will connect with database and file system. Server will get required resources from file server and database and provide it to the mobile app or web app.

Companies that uses NodeJS:-



Why are companies using NodeJs ?

- Netflix lower startup time with node.js
- NASA improved database access time with node.js
- Trello achieved quick prototyping with node.js
- Paypal decreased its loading time with node.js
- LinkedIn improved its app performance with node.js
- Walmart improved website experience with node.js
- Uber eases ride processing with node.js

Environmental Setup

Install Node.js:

Go to the official Node.js website (https://nodejs.org) and download the latest LTS (Long-Term Support) version for your operating system. LTS versions are more stable and recommended for most projects.

Install npm (Node Package Manager):

npm comes bundled with Node.js, so after installing Node.js, you'll have npm automatically.

Choose a Code Editor:

Select a code editor that suits your preferences. Some popular choices are Visual Studio Code(totally free), Atom(totally free), Sublime Text(not totally free we need to invest some amount if we want advanced feature), or WebStorm(not totally free)

Create a Project Directory:

Create a new folder on your computer to hold your Node.js project.

Initialize Your Project:

Open your terminal or command prompt, navigate to the project directory you just created, and run the following command to initialize a new Node.js project. This will generate a package.json file that will manage your project's dependencies.

Note:-What is package.json?

Keeps the details of your project like what is the version of the project, what is the name of the project, what is the git repository, which packages are there in the project. To add the package.json file, open a terminal and type npm init.

type Npm install colors. After this package is added, the node_modules folder will be created automatically. If you look in the package.json,one property named as dependencies has been added,such changes will come after the package is added.

Type npm install simple-node-logger

Package-lock.json:-It will keep the details of the package. If you delete this file, your project will work. If you run npm install command, it will be added. The package.json file should not be deleted.

Install Required Packages:-Use npm to install any Node.js packages or libraries that your project needs.

Create Your Node.js Application:-Create your Node.js application code using your preferred code editor.

Run Your Node.js Application:

Once you've written your Node.js code, save the file, and in your terminal, navigate to your project directory. Then, run your Node.js application using the following command:

```
node app.js
```

For adding more packages :

```
npm install <package name>
```

Feature of Nodejs

Single threaded

- With event looping, a single threaded architecture is followed by NodeJs and for this architecture makes NodeJs more scalable.
- In contrast to other servers, limited threads are created by them for processing the requests.
- Whereas for the event driven mechanism, the NodeJS servers reply in a non-blocking or an asynchronous manner and for this reason NodeJS becomes more scalable.
- If we compare NodeJs with other traditional servers like Apache HTTP servers, then we can say NodeJs handles a larger number of requests.
- A single threaded program is followed by NodeJS and this allows NodeJs to process a huge amount of requests.

Asynchronous

- Node.js is asynchronous by default, i.e., it operates non-blocking.

- When a client requests a server, a single thread handles the request; it checks if the request involves any database interaction. If it does not, we process the request, and the server returns the response to the client.
- The thread is ready to handle the next request.

Event-Driven Architecture

- The term event-driven means that once a task is completed something will happen. In Node.js, once an event is triggered or completed, a callback function also known as the event handler gets executed.
- Callback functions require fewer resources on the server side and also take up less memory. Due to this feature of Node JS, the application is lightweight.

Performance

- Google Chrome’s V8 JavaScript engine is the foundation of Node.js, enabling faster code execution.

- The engine compiles the JavaScript code into machine code which makes our code easier and faster to implement effectively.
- Concepts like asynchronous programming and its operation on non-blocking input-output operations make its performance high.

Scalable

- Another of the NodeJS features is scalability. This means that NodeJS can handle a large number of requests when needed.
- NodeJS can handle many requests due to its “single thread event-driven loop”. It also has a cluster module that manages load balancing for all available CPUs.

Zero Buffering

- Node.js works with data streams that are chucked data and due to this there is no or zero buffering of data when Node.js is used.

Multiple platform compatibility

- Node.js can be used on a wide range of systems, from Windows to Mac OS to Linux and even for mobile platforms.

Written in JavaScript

- JavaScript is one of the most used programming languages currently in the industry.
- Almost all web applications have JavaScript running. Most programmers already are well-equipped with JavaScript. This makes Node.js easy to understand and learn.

Import/Export in node.js

App.js:-

```
export let nm="virat kohli";
export let team="RCB";
```

index.js:-

```
import { nm } from "./app";
console.log(nm);
```

- An error will occur. Node does not support import/export. The above code will run in javascript but not in node. Browser supports import/export so above code will work in reactjs and in javascript. Node uses old javascript so does not support it.

Solution:-app.js

```
module.exports={
    nm:"virat kohli",team:"RCB",handler:function(){
        return 10;}}
```

index.js:-

```
const x=require('./app');
console.log(x);
console.log(x.handler())
```

Core Module:-

- All programming languages already have some basic features, such as features for database connectivity, features for creating files, features for calling APIs.This is called core module.
- Node has core modules for file system, buffers, HTTP etc.
- There are two type of core module.

    global module

    non global module

Global module:-

- No need to import, can use directly.

Example:-console.log()

Non global module:-

- We need to import it first

Example:-To use file system we need to import ‘fs’ module.

```
const fs=require("fs");
fs.writeFileSync("hello.txt","Advance web technology");
```

Create Server

- We need to import http module first if we want to create server in node.js
- http module handle request,response.
- createServer() function is used to create server and this function has 1 parameter(arrow function).

Example:-

```
const http=require('http');
http.createServer((req,resp)=>{
    resp.write("Hello students");//<h1>Hello Students</h1>
    resp.end();
}).listen(3001);
```

Alternate method:-

```
const http=require('http');
function dataControl(req,resp)
{
    resp.write("<h1>Hello students</h1>");
    resp.end();
}
http.createServer(dataControl).listen(3001);
```

Simple API

- Import the http module. You can use this module to create an API.
- The http module will handle the request response.

Example:-

```
const http=require('http');
http.createServer((req,resp)=>{
    resp.writeHead(200,{'Content-Type':'application/json'});
    resp.write(JSON.stringify({name:"virat kohli",city:"delhi"}));
    resp.end();//used when we fetch binary data from server
}).listen(3001);
```

- Add below code if file has more than 1 object.

data.js:-

```
data=[
    {name:"virat kohli",city:"delhi"},
    {name:"jasprit bumrah",city:"mumbai"},
    {name:"ravindra jadeja",city:"jamnagar"},
    {name:"hardik pandya",city:"baroda"}
]
module.exports=data;
```

app.js:-

```
const http=require('http');
const x=require('./data');
```

```
http.createServer((req,resp)=>{
    resp.writeHead(200,{'Content-Type':'application/json'});
    resp.write(JSON.stringify(x));
    resp.end();
}).listen(3001);
```

- In Node.js, write and writeHead are two methods provided by the http.ServerResponse class, which is used to send a response back to the client during an HTTP request.

writeHead():-Used to send response header to the client. It takes three parameters:statusCode,statusMessage and header object.

statusCode:-It indicates outcome of the request(200 for success,404 for page not found)

statusMessage:-It allows you to provide a custom status message that corresponds to the status code. If not provided, a default message will be sent based on the status code.

headers:-It is an object containing additional headers to be sent with the response.

- writeHead() method must be called before any call to write or end. It sets the response status code and headers.

write():-It is used to send the content to the client.It takes a single parameter, which is the content to be sent.You can call write multiple times to send data in chunks.

end():-The end method is used to finalize the response. After calling end, no further data can be written to the response.

Console object

- There is a built-in console object with several methods which works for printing to stdout and stderr.

console.log():-

- The console.log() method is used to print message on stdout with newline.
- Like printf() the function can take multiple arguments.

console.error():-

- The console.error() function from the console class of Node.js is used to display an error message on the console.
- It prints to stderr with a newline.

Syntax:-console.error([data][, ...args])

Parameter: This function can contain multiple parameters. The first parameter is used for the primary message and other parameters are used for substitution values.

Example:-

```
num = 220
if (num < 100) {
    console.log("Enter number greater than 100");
}
else {
    console.error("correct choice");
}
```

console.warn():-

- The console.warn() function from console class of Node.js is used to display the warning messages on the console.
- It prints to stderr with newline.
- This function is an alias of console.error() function.

Syntax:-console.warn([data][, ...args])

Parameter: This function can contains multiple parameters. The first parameter is used for the primary message and other parameters are used for substitution values.

Example:-

```
var fd = require("fs");
var path = "note.txt";
fd.open(path, "r", function(err, fd){
if (err){console.warn(err);}
else{
console.log("File open successfully" + path);
}
})
```

Note:-

Example 1:-When we install some package in react at that time sometimes we may see some warning inside terminal, developer use console.warn() to display that warning.

Example 2:-We declare one state variable in react but we don’t use that state variable inside our code at that time we will see warning message.

console.dir():-

- The console.dir() method is used to get the list of object properties of a specified object.
- These object properties also have child objects, from which you can inspect for further information.

syntax:-console.dir(object)

Parameters: This method accepts single parameter which holds the object element.

Example:-

```
let cricketer={nm:"virat",age:35}
console.dir(cricketer);
```

console.time(label)

- The console.time() method starts a timer and you can use it to track how long an operation takes.
- Each time must have and unique name (label).
- When you call console.timeEnd() with the same name, it prints the amount of time that has passed since a particular process started.

console.timeEnd(label)

- The console.timeEnd() method is used to stop a timer that was previously started by calling console.time() method.

Example:-

```
console.time('testing');
var sum=0;
for(var i=0;i<10000000;i++)
{
    sum+=i;
}
console.timeEnd('testing’);
```

console.trace():-

- The console.trace() method is an inbuilt application programming interface of the console module which is used to print stack trace messages to stderr in a newline. Similar to console.error() method.

Example:-

```
<input type="button" onClick="consoleAPI()" value="Click Me!">
 <script>
        function consoleAPI()
        {
            fn1();}
        function fn1()
        {
            fn2();}
        function fn2()
        {
            console.trace();//console.trace('msg’);
            console.error('msg');
        }
</script>
```

console.assert():-

- The console.assert() method tests whether an expression is true.
- If the expression evaluates as false it will throw an AssertionError with a message.

Example:-

```
var a=10,b=20;
console.assert(a>b,'error');
```

Package Manager

- NPM (Node Package Manager) is the default package manager for Node.js and is written entirely in JavaScript. It was initially released in January 12, 2010.
- It is a command line tool that installs, updates or uninstalls Node.js packages in your application. It is also an online repository for open-source Node.js packages.
- The node community around the world creates useful modules and publishes them as packages in this repository.
- NPM is included with Node.js installation. After you install Node.js, verify NPM installation by writing the following command in terminal or command prompt.
  ```
  npm -v
  ```
- If you have an older version of NPM then you can update it to the latest version using the following command.
  ```
  npm install npm -g
  ```

- To access NPM help, write npm help in the command prompt or terminal window.
- NPM performs the operation in two modes: global and local. In the global mode, NPM performs operations which affect all the Node.js applications on the computer whereas in the local mode, NPM performs operations for the particular local directory which affects an application in that directory only.

Install Package Locally

- Use the following command to install any third party module in your local Node.js project folder.
  npm install <package name>
- For example, the following command will install color module
  npm install colors

- All the modules installed using NPM are installed under node_modules folder. The above command will create colors folder under node_modules folder in the root folder of your project.

Add Dependency into package.json

- Use --save at the end of the install command to add dependency entry into package.json of your application.
- For example, the following command will install color module in your application and also adds dependency entry into the package.json.
  npm install colors --save

Install Package Globally

- NPM can also install packages globally so that all the node.js application on that computer can import and use the installed packages.
- NPM installs global packages into /<User>/local/lib/node_modules folder.

- Apply -g in the install command to install package globally.
- For example, the following command will install colors package globally.
  npm install -g colors

Update Package

- To update the package installed locally in your Node.js project, navigate the command prompt or terminal window path to the project folder and write the following update command.
  npm update <package name>
- The following command will update the existing colors module to the latest version.
  npm update colors

Uninstall Packages

- Use the following command to remove a local package from your project.
  npm uninstall <package name>
- The following command will uninstall colors from the application.
  npm uninstall colors

callback

- Node.js callbacks are a special type of function passed as an argument to another function.
- They're called when the function that contains the callback as an argument completes its execution, and allows the code in the callback to run in the meantime.
- Callbacks help us make asynchronous calls.
- The simplest example I can think of in JavaScript is the setTimeout() function. It's a global function that accepts two arguments. The first argument is the callback function and the second argument is a delay in milliseconds.
- The function is designed to wait the appropriate amount of time, then invoke your callback function.

Example:-

```
setTimeout(function () {
 console.log("10 seconds later...");},10000);
```

- You may have seen the above code before but just didn't realize the function you were passing in was called a callback function.

Example:-

```
function register(callback)
{
    setTimeout(()=>
    {
        console.log('register end');
        callback()
    },1000);
}
```

```
function sendEmail(callback)
{
    setTimeout(()=>{
        console.log('email end');
        callback();
    },2000);}
function login()
{
    console.log('login end');
}
function getUserData()
{
    console.log(‘Get user data');
}
```

```
function displayUserData()
{
    console.log('userdata displayed');
}
register(function(){
    sendEmail(function(){
        login();
        getUserData();
        displayUserData();
    });
});
console.log('other application work');
```

Asynchronous and synchronous

- In Synchronous operations tasks are performed one at a time.PHP is synchronous programming language.
- In Asynchronous operations second task do not wait to finish first task.Nodejs,Javascript is an Asynchronous Programming language.

Example:-

```
console.log("First task completed");
setTimeout(()=>{
    console.log("Second task completed");
},2000);
console.log("Third task completed");
```

Drawback of async:-

```
let a=10,b=0;
setTimeout(()=>
{
    b=20;
},2000);
console.log(a+b);
```

- We can handle Async data with the help of callback function, promises and async await.

Handle Asynchronous Data:-

```
let a=10,b=0;
let waitingData=new Promise((resolve,reject)=>{
    setTimeout(()=>
    {
        resolve(30);//instead b=30.
    },2000);
})
waitingData.then((data)=>{
    console.log(a+data);
})
```

Debugging

- Debugging in Node.js refers to the process of identifying and fixing issues, errors, and unexpected behaviours in your Node.js applications. It involves inspecting and analysing the code execution, variable values, and program flow to understand why the application is not functioning as expected.

Common reasons for debugging in Node.js include:

Identifying and fixing syntax errors:-Syntax errors prevent the code from running correctly, and debugging helps to locate and resolve these issues.

Finding logical errors:- Logical errors are bugs in the code that lead to incorrect results or undesired behaviour. Debugging helps pinpoint the cause of such errors and allows developers to make corrections.

Understanding runtime behaviour:- Debugging provides insights into how the code behaves during runtime, which helps in understanding complex control flows and the sequence of function calls.

Investigating exceptions and crashes:-When an unhandled exception occurs or the application crashes, debugging can help identify the root cause and fix the issue.

Inspecting variable values:- During debugging, you can inspect the values of variables at different points in the code, helping you understand why certain conditions are met or not met.

Stepping through the code:- Debugging allows developers to execute the code line-by-line or step-by-step, making it easier to understand the flow of execution and identify where issues occur.

- Node.js provides several tools and techniques for debugging, including:

Console.log() statements:- Simple console.log() statements can be used to print values to the console and observe the flow of execution.

Node.js Inspector:- The Node.js Inspector allows you to debug Node.js applications using Chrome Developer Tools. It enables you to set breakpoints inspect variables, and step through code.

Visual Studio Code (VS Code) Debugger:- VS Code offers built-in support for Node.js debugging, providing a user-friendly interface to set breakpoints, inspect variables, and control the execution flow

- Overall, debugging is a crucial skill for developers to identify and fix issues in Node.js applications, ensuring they work as intended and deliver a smooth user experience.

File System

- 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');
```

- Common use for the File System module:

    Read files

    Create files

    Update files

    Delete files

    Rename file

Read Files:-

- The fs.readFile() method is used to read files on your computer.
- Assume we have the following HTML file

```
<!DOCTYPE html>
<html lang="en">
<body>
    <h1>AWT</h1>
    <p>Advance javascript(ES6),react js,node js,mongodb,angular</p>
</body>
</html>
```

- Create a Node.js file that reads the HTML file, and return the content:

```
const http=require('http');
const fs=require('fs');
http.createServer((req,resp)=>{
    fs.readFile("one.html",function(err,data){
        resp.writeHead(200,{'Content-Type':'text/html'});
        resp.write(data);
        return resp.end();
    })
}).listen(3001);
```

Create Files

- The File System module has methods for creating new files:

  fs.appendFile()

  fs.open()

  fs.writeFile()

- The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created:

```
const fs=require('fs');
fs.appendFile("sairam.txt","Hello students!",(err)=>{
    if(!err)
    {
        console.log('file created successfully');
    }
});
```

- The fs.open() method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created:

```
const fs=require('fs');
fs.open("sairam.txt","w",(err,data)=>{
    if(!err)
    {
        console.log('file created successfully');
    }
});
```

- The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

```
const fs=require('fs');
fs.writeFile("sairam.txt","OS,AWT,CN,DS",(err)=>{
    if(!err)
    {
        console.log('file created successfully');
    }
});
```

Update Files:

- The File System module has methods for updating files:
  - fs.appendFile()
  - fs.writeFile()
- The fs.appendFile() method appends the specified content at the end of the specified file:

```
const fs=require('fs');
fs.appendFile("sairam.txt","Text has been added",(err)=>{
    if(!err)
    {
        console.log('file updated successfully');}});
```

Delete Files

- To delete a file with the File System module, use the fs.unlink() method.
- The fs.unlink() method deletes the specified file:

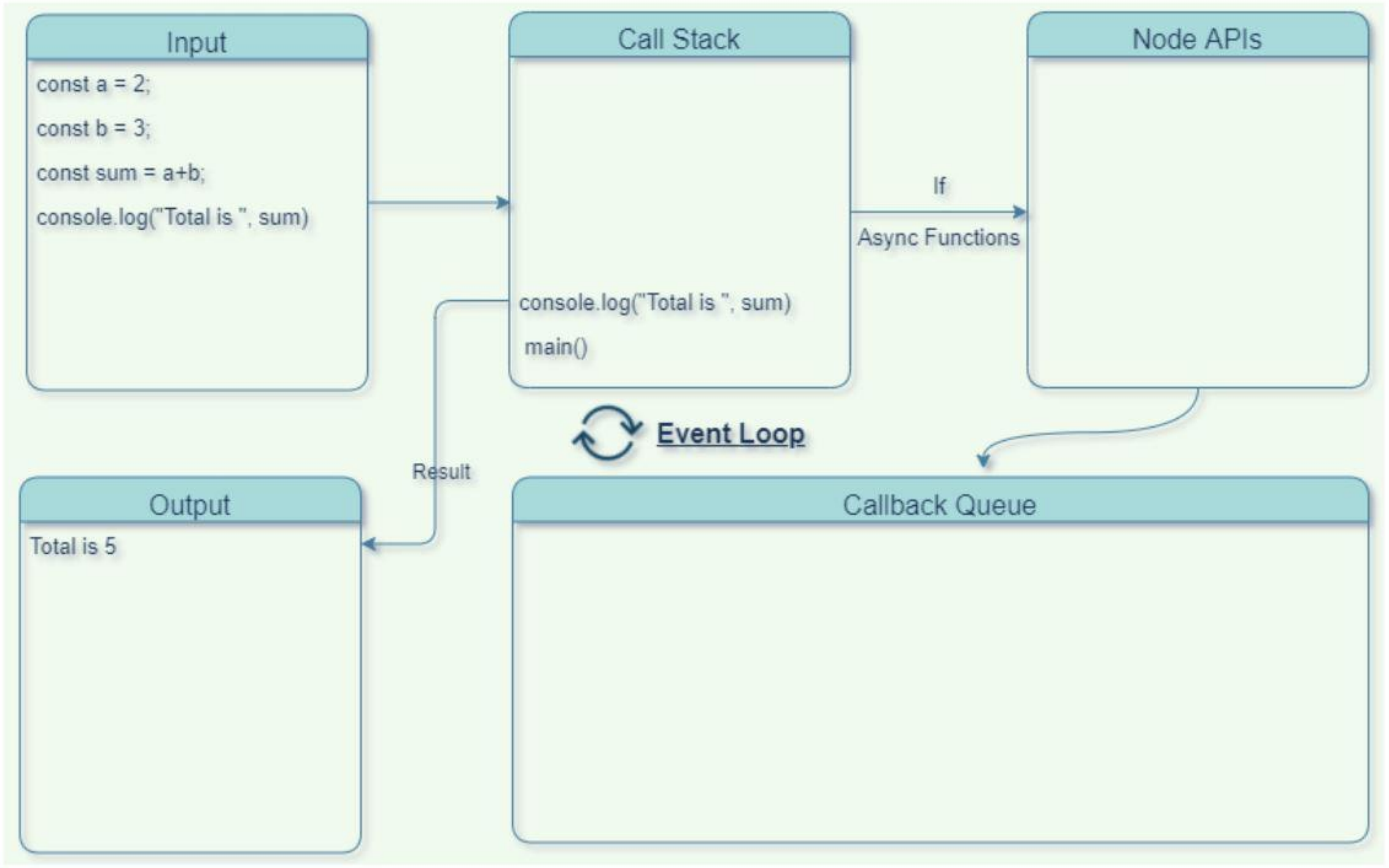
```
const fs=require('fs’);
fs.unlink("sairam.txt",(err)=>{
    if(!err){
      console.log('file deleted');
    }
});
```

Rename Files

- To rename a file with the File System module, use the fs.rename() method.
- The fs.rename() method renames the specified file:

```
const fs=require('fs');
fs.rename("sairam.txt","subject.txt",(err)=>{
    if(!err)
    {
        console.log('file renamed');
    }
});
```

Event Loop



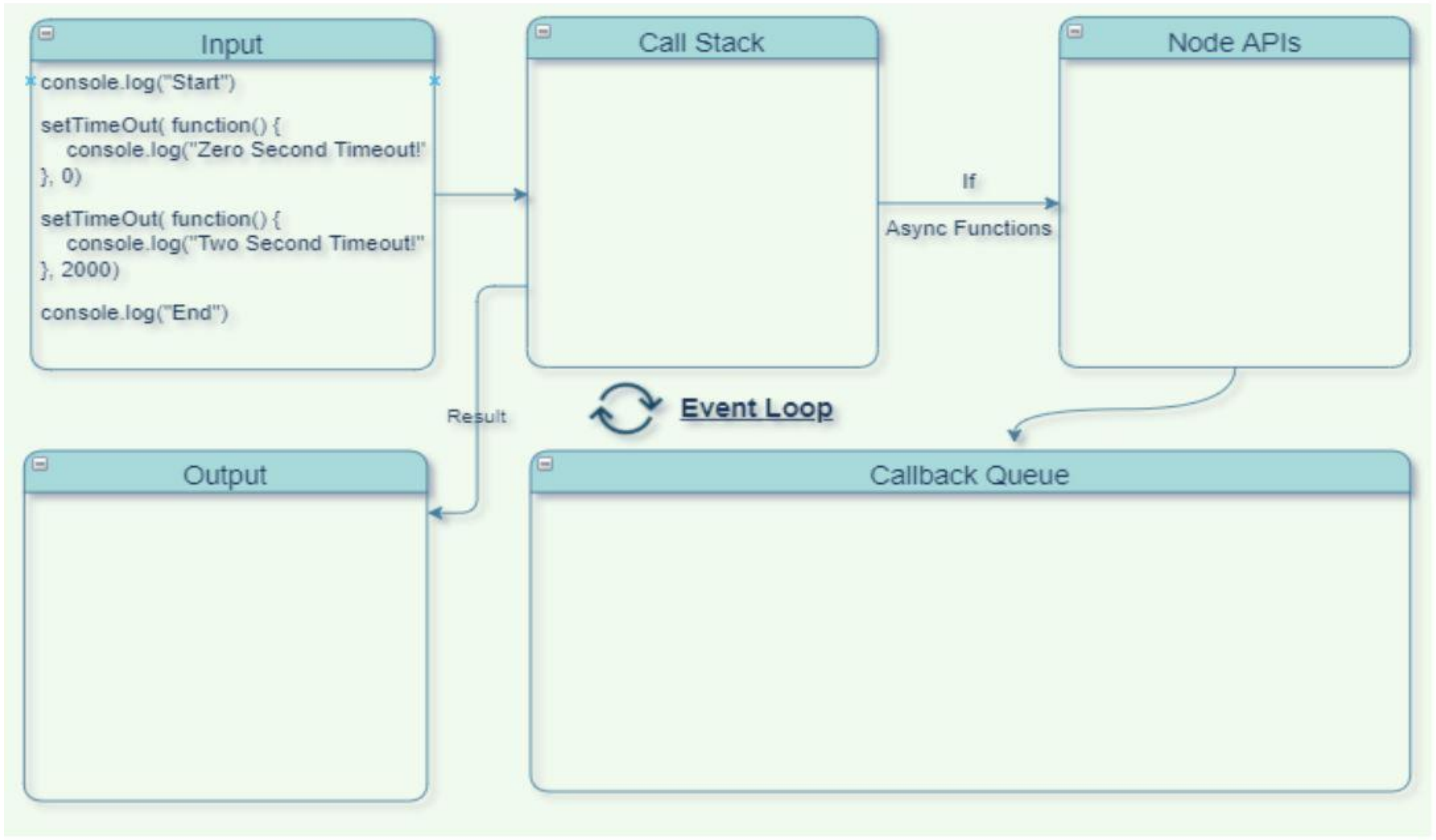
```
const a = 2;
const b = 3;
const sum = a+b;
console.log("Total is ", sum)
```

```
console.log("Total is ", sum)
main()
```

```
Total is 5
```

- Node handles requests using an Event loop inside the NodeJS environment.
- An event loop is an event-listener which functions inside the NodeJS environment and is always ready to listen, process, and output for an event.
- An event can be anything from a mouse click to a keypress or a timeout.
- Now let's see how NodeJS event loops can execute a simple synchronous program using a Nodejs event loop diagram. Then we'll examine how Node executes the program line by line.
- In the top-left corner, you have a Node file that is going to get executed. At the bottom left, you have an output terminal for the program. Then you have Call stack, Node APIs and Callback queue. All these together constitute the NodeJS environment.

- For synchronous programming, you only need to focus on the call stack. This is the only part of the NodeJS environment that will be working in this case.
- A callback stack is a data structure that you use to keep track of the execution of all functions that will run inside the program. This data structure has only one open end to add or remove top items.
- When the program starts executing, it first gets wrapped inside an anonymous main() function. This is automatically defined by NodeJS. So main() gets pushed first to the callback stack.
- Next, the variables a and b are created and their sum is stored in a variable sum. All these values are stored in memory.
- Now, the console.log() is a function that is called and pushed inside the callback stack. It gets executed and you can see the output on the terminal screen.
- After this function gets executed, it is removed from the callback stack. Then the main() is also removed as nothing is left to be called from the program. This is how a synchronous program gets executed.



```
console.log("Start")

setTimeOut( function() {
    console.log("Zero Second Timeout!"
}, 0)

setTimeOut( function() {
    console.log("Two Second Timeout!"
}, 2000)

console.log("End")
```

- Now, let's see how asynchronous functions or programs get executed inside NodeJS. We need the callback stack, Node APIs and callback queue all together to process an asynchronous function.
- As usual, when the program starts executing, first the main() function gets added to the callback stack. Then console.log("Start") is called and added to the callback stack. After processing, the output is visible on the terminal and then it gets removed from the callback stack.
- Now the next is the setTimeOut(...Zero...) function which gets added to the callback stack.
- As this is an asynchronous function, it will not get processed in the callback stack. It is then added from the callback stack to the Node APIs where an event is registered and a callback function is set to get processed in the background.

- Next is the setTimeOut(...Two..) which also gets added to the Node API from the callback stack as it is an asynchronous function. Then another callback function gets set to be processed after a 2 second timeout in the background.
- This is called non-blocking behaviour where all the synchronous functions are processed and executed first and asynchronous functions are processed in the background while waiting their turn to get executed.
- Next, the console.log("End") function is called at last in the callback stack and gets processed here. You can see the output on the terminal. Now, all the synchronous functions are processed, and main() is removed from the callback stack.
- In the background, all the asynchronous functions get processed and their callbacks are stored in the callback queue. The one which is processed first will be added first in the queue for execution in the callback stack.

- Asynchronous functions cannot run inside a callback stack until it gets emptied. That means that after main() is removed from call stack, only then can all asynchronous functions start executing.
- Now, one by one they are pushed to the callback stack using the event loop and finally get executed. Each of the callback functions will print the value with the console.log() function getting called each time. At last, these are also removed after being executed and now the callback stack is empty.
- This is how NodeJS will execute synchronous and asynchronous funtions inside the environment and how the event loop manages to call asynchronous functions.

Event and Event emitter

- Everything in Node is event based.
- All most all programming languages has events, but events are mostly used in javascript and HTML.
- node has one module called events, we don't need to install them. We need to import it just like we import http module, file system module.
- The one who generates the event is called emitter. A button in HTML is called an emitter.

Example:-Display API call count(store API call count in database)

```
const express=require('express');
const EventEmitter=require('events');//EventEmitter is class
const app=express();
const event=new EventEmitter();
```

```
var count=0;
event.on("countAPI",()=>{
    count++;
    console.log(`API calls ${count}`);
})
app.get("/",(req,resp)=>{
    resp.send("Api called");
    event.emit("countAPI");//generate an event
}).listen(5000);
```

Stream

- Streams are one of the fundamental concepts of Node.js. Streams are a type of data-handling methods and are used to read or write input into output sequentially.
- Streams are used to handle reading/writing files or exchanging information in an efficient way.
- Streams are objects in Node.js that lets the user read data from a source or write data to a destination in a continuous manner.
- There are namely four types of streams in Node.js.

  Writable: We can write data to these streams.

  Readable: We can read data from these streams.

  Duplex: Streams that are both, Writable as well as Readable.

  Transform: Streams that can modify or transform the data as it is written and read.

- Each type of stream is an EventEmitter instance and throws several events at different instance of times. Some commonly used events are

data:-This event is fired when there is data available to read

end:-This event is fired when there is no more data to read

error:-This event is fired when there is any error receiving or writing data

finish:-This event is fired when all the data has been flushed to underlying system

Example:-

app.js:-

```
const x=require('fs');
for(let i=1;i<10000;i++)
{ x.writeFileSync('subject.txt',`hello world ${i}\n`,{flag:'a'});}
```

index.js:-

```
const {createReadStream}=require('fs');
const stream=createReadStream('subject.txt');
stream.on('data',(result)=>
{
    console.log(result);
});
```

Timer

- A timer is used in JavaScript to execute a task or function at a specific time.
- The timer is essentially used to delay the execution of the program or to execute the JavaScript code at regular intervals.
- Advertisement banners on websites, which change every 2-3 seconds, are the best example of a timer. These advertising banners are rotated at regular intervals on websites such as Flipkart. To change them, you set a time interval.

setTimeout():-

- setTimeout() method is used to execute a piece of code after a specified time. It takes the first argument as a function and the second argument as time in milliseconds. The code will be executed after the time that is passed as the argument.

Example:-

```
setTimeout(function() {
   console.log('Hello World!');
}, 2000);
console.log('Executed before function’);
```

clearTimeout():-

- The clearTimeout() method deactivates a timer that was previously set with the setTimeout() method.
- The ID value returned by setTimeout() is passed to the clearTimeout() method as a parameter.

Syntax:-clearTimeout(id_of_settimeout)

Example:-

```
function welcome () {
   console.log("Welcome to Knowledgehut!");
  }
var id1 = setTimeout(welcome,1000);
clearTimeout(id1);
```

setInterval():-

- The setInterval() method gives the privilege to keep executing a piece of code after a particular period of time. It takes the first argument as the callback function and the second argument is the time in milliseconds.

Example:-

```
setInterval(function()
{
        console.log('Hello world');
},2000);
```

clearInterval():-

- This method is used for cancelling the timer method that was scheduled using the setInterval() method.

Example:-

```
const timer = setInterval(function A() {
        console.log("Hello Geek!");
}, 1000);
```

```
setTimeout(function () {
    clearInterval(timer);
    console.log("Interval Timer cleared")
}, 3000);
```

setImmediate()

- The setImmediate() function runs code after the current event loop cycle is completed.
- setImmediate() is similar to setTimeout() with a 0ms time delay.
- To put it another way, the setImmediate() method will execute code after each statement in the script is executed.

```
console.log('first task');
setImmediate(function()
{
    console.log('set immediate called');
})
console.log('second task');
for(let i=1;i<10;i++)
{
    console.log(i);
}
```

clearImmediate()

- The clearImmediate() function stops the Immediate objected returned by the setImmediate() function.

```
console.log('first task');
const myImmediate=setImmediate(function()
{
    console.log('set immediate called');
})
console.log('second task');
clearImmediate(myImmediate);
for(let i=1;i<10;i++)
{
    console.log(i);
}
```

Buffer

- Buffer is one container in memory.
- It is similar to array.
- A buffer will have elements like an array. This element will also have an index number.
- When you allocate memory to the buffer, all the elements will be filled with zero.
- The size of the each element in the buffer will be 8 bits.
- After allocating the size of the buffer, it will not change.
- A buffer is a data structure like an array designed to work with binary data.
- When you create the buffer, i.e. allocate the memory, it will have all zeros, then data can be added to it.

```
const {Buffer}=require('buffer');
const memoryContainer=Buffer.alloc(4);// 4 bytes or 32 bits
//console.log(memoryContainer);
//console.log(memoryContainer[0]);
memoryContainer[0]=0xF4;//11110100
//console.log(memoryContainer[0]);
memoryContainer[1]=0x34;//00110100
memoryContainer[2]=0xb6;//10110110
memoryContainer[3]=0xff;//11111111
//console.log(memoryContainer[3]);
//console.log(memoryContainer.toString("hex"));
//The toString() method returns the buffer object according to the specified encoding.
```

```
//const buff=Buffer.from([0x48,0x69,0x21]);
//console.log(buff);
//from() method creates a new buffer filled with the specified string, array, or buffer.
//console.log(buff.toString("utf-8"));
//const buff=Buffer.from("486921","hex");
//console.log(buff);
//console.log(buff.toString("utf-8"));
const buff=Buffer.from("Hi!","utf-8");
console.log(buff)
```

```
//var buf = Buffer.from('abcdef’);
//buf.write('qq',2);
//console.log(buf);
//console.log(buf.toString());
var buf = Buffer.from('abcdef');
console.log(Buffer.isBuffer(buf));
var buf = Buffer.from('abc');
console.log(buf.length);
```

Marwadi University

Thank You