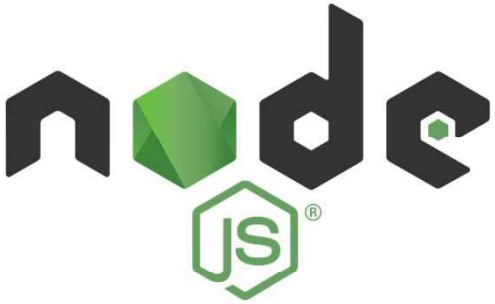
22ITC08

FULL STACK DEVELOPMENT

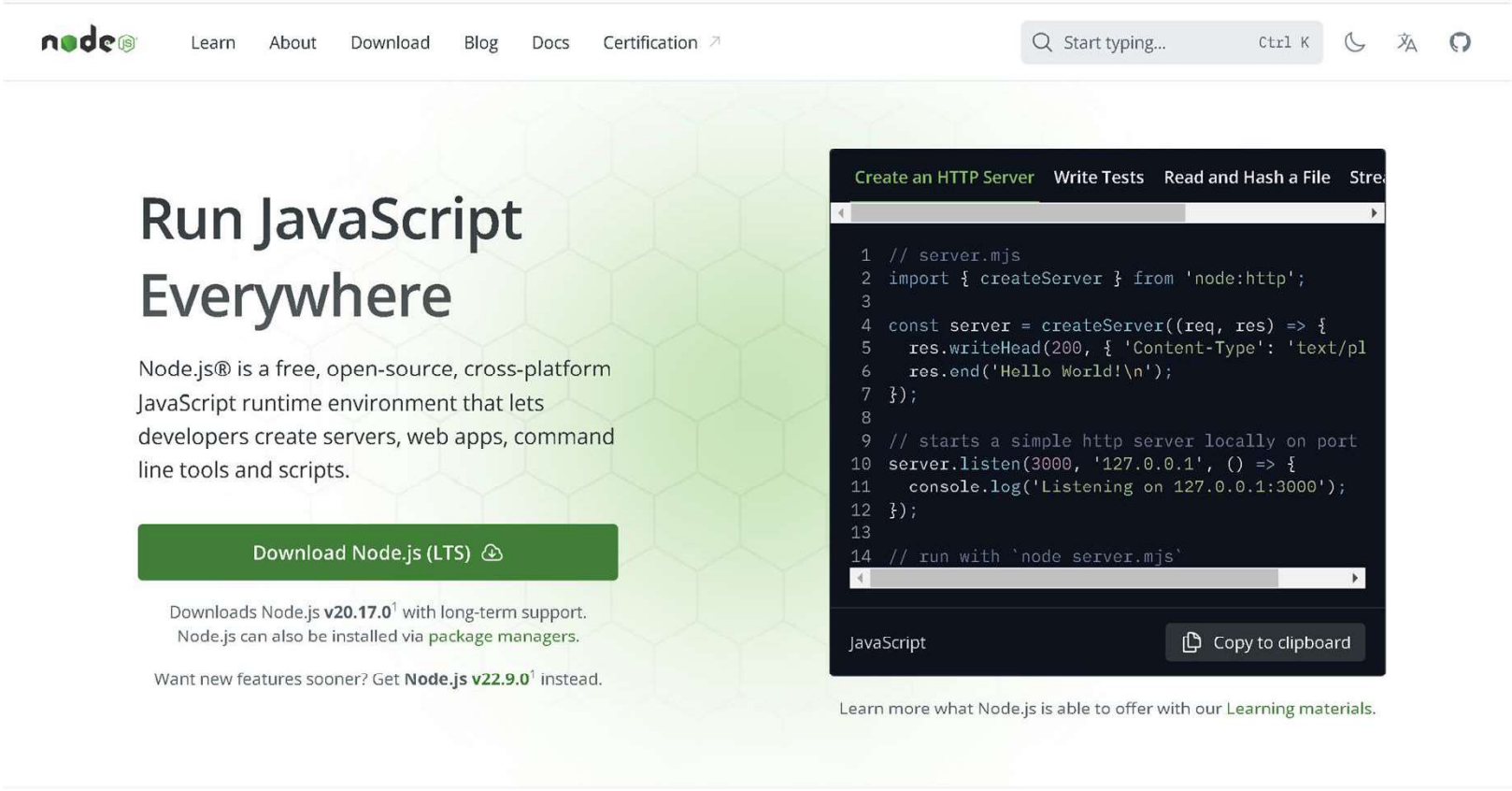
UNIT-IV

Node JS



Node JS - https://nodejs.org/en/

Node.js® is an open-source, cross-platform JavaScript runtime environment.



Node JS

- Node.js is an **open-source and cross-platform runtime environment built on Chrome's V8 JavaScript engine** for executing JavaScript code outside of a browser.
- **NodeJS is not a framework** and it's **not a programming language.**

To check your node version:

$ node --version / $ node -v

To run JS file:

$ node app.js

Node JS Modules

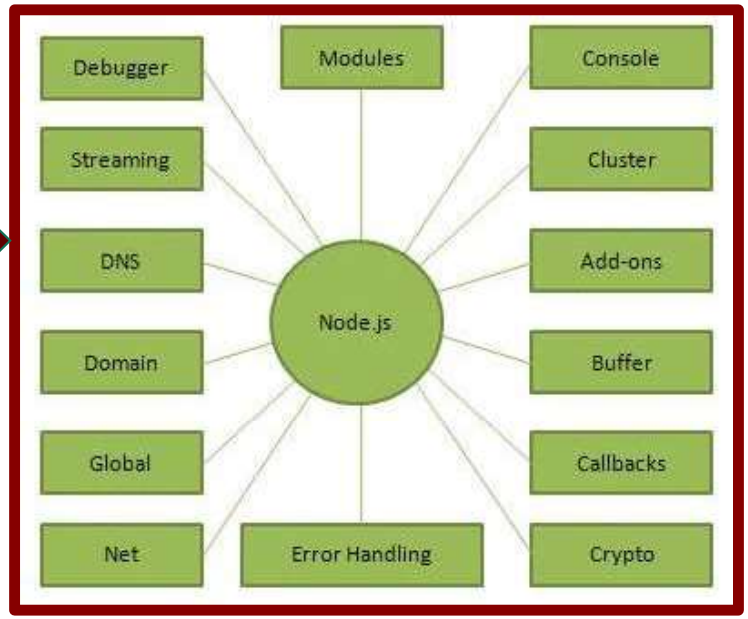
- In NodeJS, **Modules are the blocks of encapsulated code** that **communicates with an external application** on the basis of their related functionality.

Types of Modules in NodeJS:

1) Core/Built-in Modules

2) Local Modules

3) Third-party Modules



NodeJS Core Modules

- Built-in modules of node.js that are part of NodeJS and come with the Node.js installation process are known as core modules.
- To load/include this module in our program, we use the require function.

const module = require('module_name')

Node.js = Runtime Environment + JavaScript Library

Node JS Documentation

https://nodejs.org/docs/latest-v17.x/api/

NodeJS Local Modules

- Local modules are created by us **locally in our Node.js application.** These modules are included in our program in the same way as we include the built in module.

Code for creating local modules and exporting:

```
exports.add=function(x,y)
{
      return x+y;
};
```

Code for including local modules:

```
let sum = require('./sum')

console.log("Sum of 10 and 20 is ", sum.add(10, 20))
```

NodeJS Local Modules

execution.js

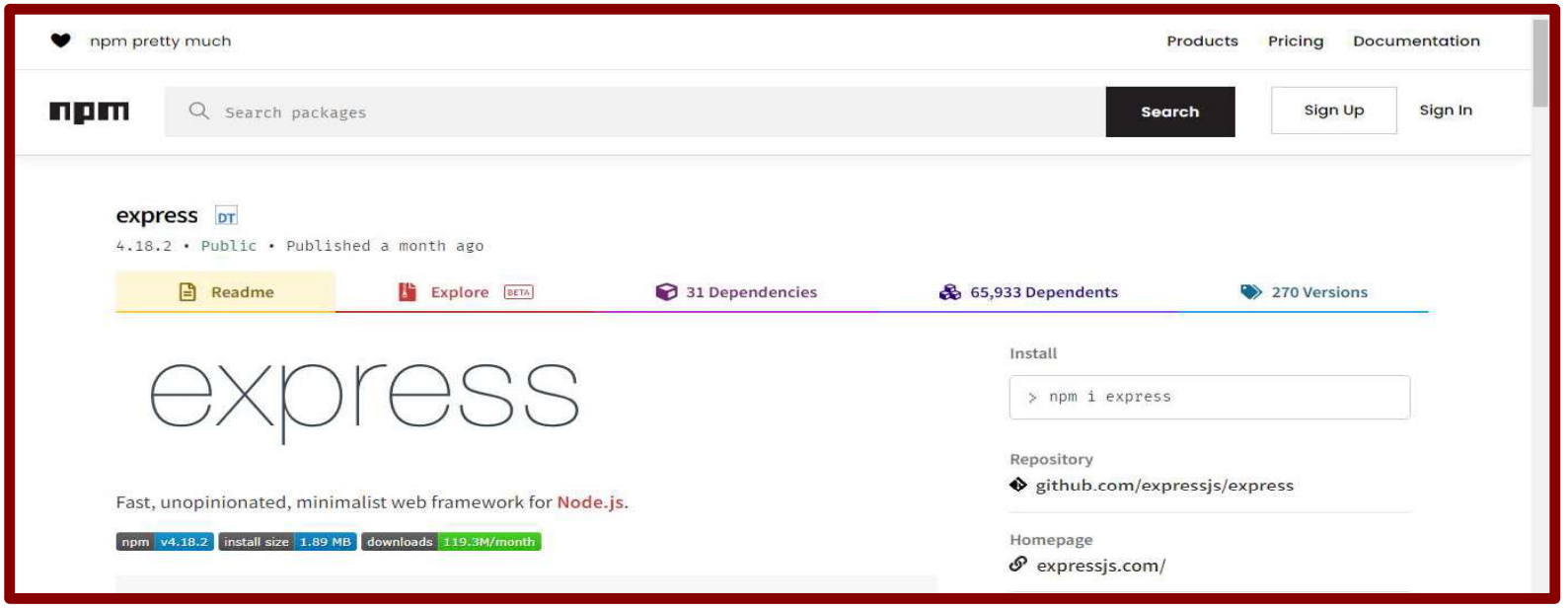
```
var cal=require('./mathcal');
var x = 10;
var y = 5;
console.log(" Add : " + cal.add(x,y));
console.log(" Sub : " + cal.sub(x,y));
console.log(" Mul : " + cal.mul(x,y));
console.log(" Div : " + cal.div(x,y));
```

mathcal.js

```
exports.add=function(x,y)
{
    return x+y;
};
exports.sub=function(x,y)
{
    return x-y;
};
exports.mul=function(x,y)
{
    return x*y;
};
exports.div=function(x,y)
{
    return x/y;
};
```

NodeJS Third Party Modules

- **Modules** that are available online and are **installed using the npm are called third party modules.**
- Examples of third party modules are **express, mongoose, Amazon Alexa etc.**



Export Module in Node.js

- The **module.exports is a special object** which is included in every JavaScript file in the Node.js application by default.
- The **module is a variable** that represents the current module.
- The **exports is an object** that will be exposed as a module. So, whatever you assign to module.exports will be exposed as a module.

Export Module in Node.js

Export Literals

As mentioned above, exports is an object. So it exposes whatever you assigned to it as a module.

Message.js

```
module.exports = 'Hello world';
```

app.js

```
var msg = require('./Messages.js');

console.log(msg);
```

```
C:\> node app.js
Hello World
```

Export Module in Node.js

Export Object

The exports is an object. So, you can attach properties or methods to it.

Message.js

```
exports.SimpleMessage = 'Hello world';

//or

module.exports.SimpleMessage = 'Hello world';
```

app.js

```
var msg = require('./Messages.js');

console.log(msg.SimpleMessage);
```

```
C:\> node app.js
Hello World
```

Export Module in Node.js

Export Object

- **Use exports when you want to add properties to the export object**.
- **Use module.exports when you want to export a single object or function**, or if you need to completely replace the exports object.

```
// myModule.js
exports.SimpleMessage = "CBIT"; // Adds a property to the exports object
exports.AnotherMessage = "Hello"; // Another property added
```

```
// main.js
const myModule = require('./myModule');
console.log(myModule.SimpleMessage); // Outputs: CBIT
console.log(myModule.AnotherMessage); // Outputs: Hello
```

```
// myModule.js
module.exports = {
                SimpleMessage: "CBIT", // Sets the entire export object
                AnotherMessage: "Hello"
                };
```

```
// main.js
const myModule = require('./myModule');
console.log(myModule.SimpleMessage); // Outputs: CBIT
console.log(myModule.AnotherMessage); // Outputs: Hello
```

Export Module in Node.js

- In the same way as above, you can **expose an object with function**. The following example **exposes an object with the log function as a module**.

Log.js

```
module.exports.log = function (msg) {
    console.log(msg);
};
```

app.js

```
var msg = require('./Log.js');

msg.log('Hello World');
```

```
C:\> node app.js
Hello World
```

Export Module in Node.js

- You can also **attach an object to module.exports**, as shown below.

data.js

```
module.exports = {
    firstName: 'James',
    lastName: 'Bond'
}
```

app.js

```
var person = require('./data.js');
console.log(person.firstName + ' ' + person.lastName);
```

```
C:\> node app.js
James Bond
```

Export Module in Node.js

Export Function

You can **attach an anonymous function to exports object** as shown below

Log.js

```
module.exports = function (msg) {
    console.log(msg);
};
```

app.js

```
var msg = require('./Log.js');

msg('Hello World');
```

```
C:\> node app.js
Hello World
```

Export Module in Node.js

Export Function as a Class

In JavaScript, **a function can be treated like a class (ES5)**. The following example exposes a function that can be used like a class.

Person.js

```
module.exports = function (firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.fullName = function () {
        return this.firstName + ' ' + this.lastName;
    }
}
```

app.js

```
var person = require('./Person.js');

var person1 = new person('James', 'Bond');

console.log(person1.fullName());
```

```
C:\> node app.js
James Bond
```

Export Module in Node.js

Load Module from the Separate Folder

- Use the **full path of a module** file where you have exported it using module.exports.

app.js

```
var log = require('./utility/log.js');
```

app.js

```
var log = require('./utility');
```

NPM - Node Package Manager

- **NPM is the world's largest Software Registry.** The registry contains over 800,000 code packages.
- **Node Package Manager (NPM) is a command line tool** that installs, updates or uninstalls Node.js packages in your application.
- Official website: https://www.npmjs.com
- **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. **C:\> npm –v** **10.8.3**

NPM - Node Package Manager

- If you have an **older version of NPM then you can update** it to the latest version using the following command.

C:\> npm install npm –g

- **To access NPM help**, write **npm help** in the command prompt or terminal window.

C:\> npm help

NPM - Node Package Manager

When we install a package using npm you can perform two types of installation:

Local Installation

Use the following command to install any third party module in your local Node.js project folder.

C:\>npm install <package name>

C:\MyNodeProj> npm install express

All the modules installed using NPM are installed under **node_modules** folder. ~~**C:\MyNodeProj> npm install express --save**~~

NPM - Node Package Manager

Global Installation

- **NPM can also install packages globally** so that all the node.js application on that computer can import and use the installed packages.
- **Global modules are installed in** the standard system in root location in **system directory /usr/local/lib/node_modules** project directory.
- **To Print location on your system** where all the global modules are installed.

$ npm root –g

C:\Users\Admin\AppData\Roaming\npm\node_modules

Node JS

To List all the Global Packages in the system:

$ npm list -g

Update Package

C:\MyNodeProj> npm update <package name>

Uninstall Packages

C:\>npm uninstall <package name>

NodeJS Web Server

- **To access web pages of any web application, you need a web server.**
- The **web server** will **handle all the http requests** for the web application.
- **IIS is a web server for ASP.NET web applications** and **Apache is a web server for PHP or Java web applications**.
- **Node.js provides capabilities to create your own web server** which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but **it is recommended to use Node.js web server.**

NodeJS Web Server

- **HTTP stands for Hyper Text Transfer Protocol**

- **require**

We use the **require** directive to load Node JS modules.

- **Create Server**

The HTTP module use the createServer() method to create an HTTP server.

NodeJS Web Server

HTTP Header If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type.

The first argument of the **res.writeHead()** method is the **status code, 200 means that all is OK**, the second argument is an **object containing the response headers.**

Port Number

HTTP server that listens to server ports and gives a response back to the client.

NodeJS Web Server

```
//1 - Import Node.js core module
var http = require('http');
// 2 - creating server
http.createServer(function(req,res)
{
    //handle incomming requests here..
    res.writeHead(200,{'Content-type':'text/plain'});
    res.write("Welcome to CBIT");
    res.end("Thank You");
})
//3 - listen for any incoming requests
server.listen(5000);
console.log('Node.js web server at port 5000 is running..')
```

NodeJS Web Server-Handle HTTP Request

```
// Import Node.js core module
var http = require('http');
//create web server
var server = http.createServer(function (req, res)
    {
    if (req.url == '/') { //check the URL of the current request

        // set response header
        res.writeHead(200, { 'Content-Type': 'text/html' });

        // set response content
        res.write('<html><body><p>This is home Page.</p></body></html>');
        res.end();

    }
```

NodeJS Web Server

```
    else if (req.url == "/student")
    {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/admin")
    {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is admin Page.</p></body></html>');
        res.end();

    }
    else
        res.end('Invalid Request!');

});
//6 - listen for any incoming requests
server.listen(5000);

console.log('Node.js web server at port 5000 is running..')
```

NodeJS File System - fs

- Node.js includes **fs** module to access physical file system. The fs module is responsible for all the asynchronous or synchronous file I/O operations.
- **To include the File System module, use the require() method**:

```
var fs = require('fs');
```

- Common use for the File System module:

1. Read files
2. Create files
3. Update files
4. Delete files
5. Rename files

NodeJS File System - Synchronous

- All the fs operations can be performed in a synchronous as well as in an asynchronous approach depending on the user requirements.
- **Synchronous methods:** Synchronous functions block the execution of the program until the file operation is performed. These functions are also called **blocking functions.**

fs.readFileSync()

NodeJS File System - Asynchronous

- **Asynchronous functions do not block the execution of the program** and each command is executed after the previous command even if the previous command has not computed the result.
- The previous command runs in the background and loads the result once it has finished processing.

fs.readFile(fileName [,options], callback)

Synchronous Vs. Asynchronous

```
var fs = require("fs");

// Synchronous read
console.log("Start Synchronous read method:");
var data = fs.readFileSync('sample.txt');
console.log("Data in the file is - " + data.toString());
console.log("End Synchronous read method:");
```

```
var fs = require("fs");

// Asynchronous read
console.log("Asynchronous read method:");
fs.readFile('sample.txt', function (err, data) {
        if (err)
        {
           return console.error(err);
        }
console.log("Data in the file is - " + data.toString());
});
console.log("I am outside the function");
```

```
❯ node index.js
Start Synchronous read method:
Data in the file is - Welcome to CBIT
End Synchronous read method:
```

```
~/projects/node-xvlwyp
❯ node index.js
Asynchronous read method:
I am outside the function
Data in the file is - Welcome to CBIT
```

NodeJS File System - fs

- Use **fs.readFile()** method to read the physical file asynchronously.

fs.readFile(fileName [,options], callback)

Parameter Description:

- **filename:** Full path and name of the file as a string.
- **options:** The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
- **callback:** A function with two parameters err and data. This will get called when readFile operation completes.

NodeJS File System - fs

CBIT.html

```
<html>
    <body>
        <h1>Welcome to CBIT</h1>
        <p>IT3-Full Stack Development</p>
    </body>
</html>
```

index.js

```
var fs = require('fs');

var data = fs.readFileSync('CBIT.html', 'utf8');
console.log(data);
```

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

NodeJS File System - fs

- Use **fs.writeFile()** method to write data to a file. If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

fs.writeFile(filename, data[, options], callback)

Parameter Description:

- **filename:** Full path and name of the file as a string.
- **Data:** The content to be written in a file.
- **options:** The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".
- **callback:** A function with two parameters err and data. This will get called when write operation completes.

NodeJS File System - fs

```
var fs = require("fs");

console.log("writing into existing file");
fs.writeFile('input.txt', 'Welcome To CBIT', function(err) {
    if (err)
    {
    return console.error(err);
    }
console.log("Data written successfully!");
console.log("Let's read newly written data");

fs.readFile('input.txt', function (err, data) {
    if (err)
    {
        return console.error(err);
    }
    console.log("Asynchronous read: " + data.toString());
});
});
```

NodeJS File System - fs

- The **fs.appendFile()** method is used to synchronously append the data to the file.

fs.appendFile(filepath, data, options, callback);

fs.appendFileSync(filepath, data, options);

Parameters:

- **filepath:** It is a String that specifies the file path.
- **data:** It is mandatory and it contains the data that you append to the file.
- **options:** It is an optional parameter that specifies the encoding/mode/flag.
- **Callback:** Function is mandatory and is called when appending data to file is completed.

NodeJS File System - fs

```
var fs = require('fs');

var data = "\nLearn Node.js";

// Append data to file
fs.appendFile('input.txt', data, 'utf8',function(err) {
        if (err)
            throw err;

        // If no error
        console.log("Data is appended to file successfully.")
});
```

NodeJS File System - fs

- The **fs.open()** method is used to create, read, or write a file.

fs.open(path, flags, mode, callback)

Parameters:

path: It holds the name of the file to read or the entire path if stored at other locations.

flags: Flags indicate the behavior of the file to be opened.

mode: Sets the mode of file i.e. **r-read**, **w-write**, **r+ -readwrite (Default)**.

NodeJS File System - fs

```
var fs = require("fs");

// Asynchronous - Opening File
console.log("opening file!");
fs.open('Hyderabad.txt', 'r+', function(err, fd) {
    if (err)
    {
    return console.error(err);
    }
console.log("File open successfully");
});
```

NodeJS File System - fs

Flag	Description
r	Open file for reading. An exception occurs if the file does not exist.
r+	Open file for reading and writing. An exception occurs if the file does not exist.
rs	Open file for reading in synchronous mode.
rs+	Open file for reading and writing, telling the OS to open it synchronously. See notes for 'rs' about using this with caution.
w	Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
wx	Like 'w' but fails if path exists.
w+	Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
wx+	Like 'w+' but fails if path exists.
a	Open file for appending. The file is created if it does not exist.
ax	Like 'a' but fails if path exists.
a+	Open file for reading and appending. The file is created if it does not exist.
ax+	Like 'a+' but fails if path exists.

NodeJS File System - fs

- **Delete a File:** The **fs.unlink()** method is used to remove a file or symbolic link from the file system.
- This function does not work on directories, therefore it is recommended to use **fs.rmdir() to remove a directory**.

fs.unlink(path, callback);

NodeJS File System - fs

```
var fs = require("fs");

console.log("deleting an existing file");
fs.unlink('input.txt', function(err) {
     if (err)
     {
     return console.error(err);
     }
console.log("File deleted successfully!");
});
```

NodeJS Event Emitter class

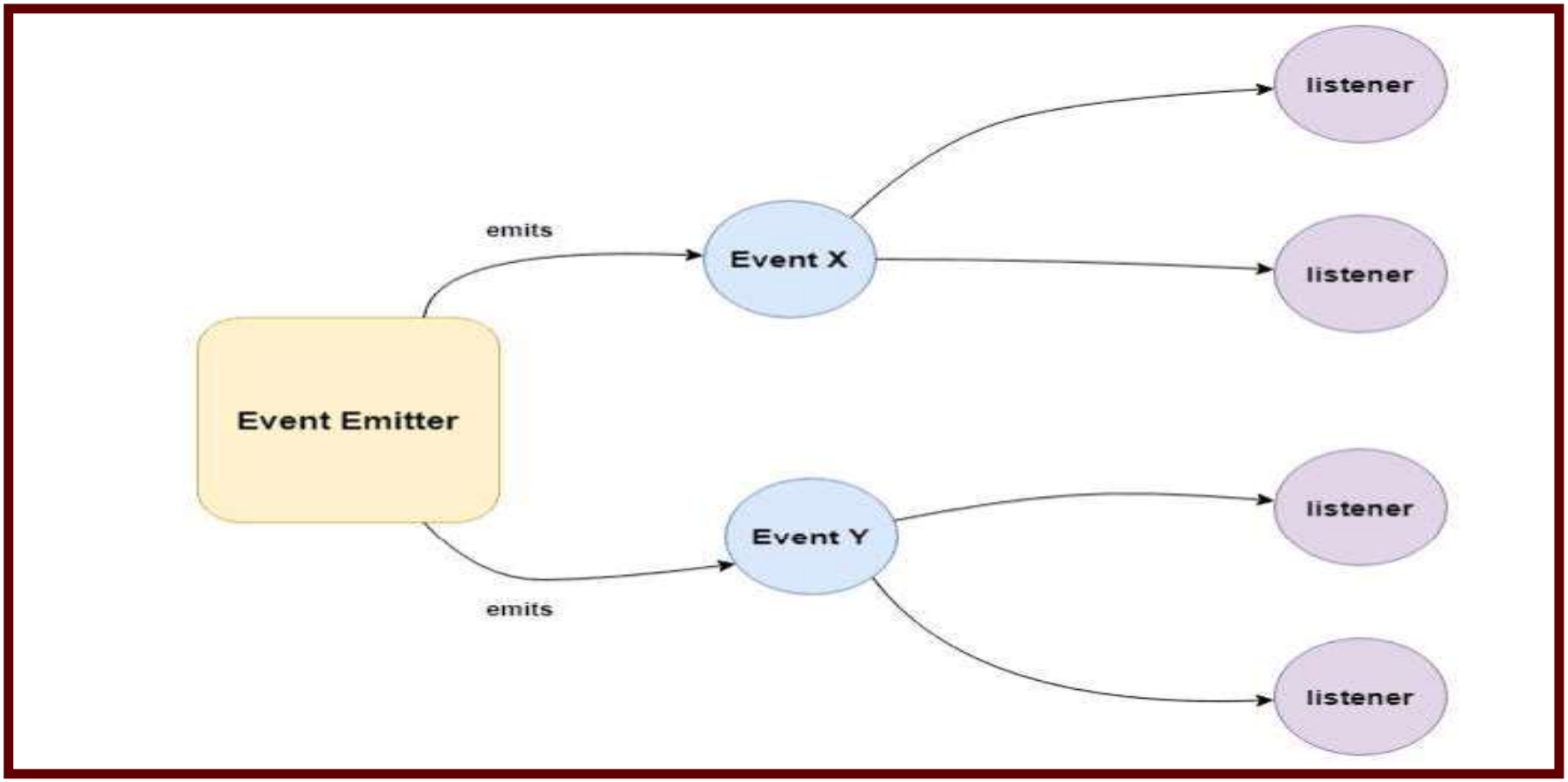
- **Every action on a computer is an event**. Like when a connection is made or a file is opened.
- Node.js allows us to **create and handle custom events easily by using events module**. Event module includes **EventEmitter class** which can be used to raise and handle custom events.

```
const EventEmitter = require('events');
```

```
eventEmitter.addListener(event, listener)
```

```
eventEmitter.on(event, listener)
```

NodeJS Event Emitter class



NodeJS Event Emitter class

```
// get the reference of EventEmitter class of events module
var events = require('events');
//create an object of EventEmitter class by using above reference
var eventEmitter = new events.EventEmitter();

// Registering/Subscribe to doorOpen Event
eventEmitter.on('doorOpen', (msg) => {
  console.log("Ring Ring Ring");
  console.log(msg);
});
// Triggering doorOpen
eventEmitter.emit('doorOpen','IT3 IS BEST');
```

NodeJS Event Emitter class

```
var emitter = require('events').EventEmitter;

var em = new emitter();

//Subscribe FirstEvent
em.addListener('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

//Subscribe SecondEvent
em.on('SecondEvent', function (data) {
    console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');

// Raising SecondEvent
em.emit('SecondEvent', 'This is my second Node.js event emitter example.');
```

NodeJS Event Emitter class

Common Patterns for EventEmitters

There are **two common patterns** that can be used to raise and bind an event using EventEmitter class in Node.js.

Return EventEmitter from a function

In this pattern, function returns an EventEmitter object, which was used to emit events inside a function. This EventEmitter object can be used to subscribe for the events.

Extend the EventEmitter class

In this pattern, we can extend the constructor function from EventEmitter class to emit the events.

EventEmitter class using **util.inherits()** method of utility module.

NodeJS Returning Event Emitter.

```
var emitter = require('events').EventEmitter;
function LoopProcessor(num)
{
    var e = new emitter();

            setTimeout(function ()
            {
             for (var i = 1; i <= num; i++)
                        {
                    e.emit('BeforeProcess', i);
                    console.log('Processing number:' + i);
                    e.emit('AfterProcess', i);
                    }
            } , 2000)
             return e;
}
var lp = LoopProcessor(3);
lp.on('BeforeProcess', function (data) {
    console.log('About to start the process for ' + data);
});

lp.on('AfterProcess', function (data) {
    console.log('Completed processing ' + data);
});
```

NodeJS Inheriting Events

```
var emitter = require('events').EventEmitter;
var util = require('util');
function LoopProcessor(num)
{
    var me = this;
    setTimeout(function ()
    {
            for (var i = 1; i <= num; i++)
                {
                me.emit('BeforeProcess', i);
                console.log('Processing number:' + i);
                me.emit('AfterProcess', i);
                 }
    } , 2000)
          return this;
}
util.inherits(LoopProcessor, emitter)
var lp = new LoopProcessor(3);
lp.on('BeforeProcess', function (data) {
    console.log('About to start the process for ' + data);
    });

lp.on('AfterProcess', function (data) {
    console.log('Completed processing ' + data);
});
```

20ADC07

FULL STACK DEVELOPMENT

UNIT-IV

Express JS

What is Express JS?

Fast

https://expressjs.com/

unopinionated,

minimalist

web framework

for Node.js

```
$ npm install express
```

Template Engine in Express JS

- **Template engine** helps us to create an HTML template with minimal code. Also, it can inject data into HTML template at client side and produce the final HTML.
- **A Template Engine** enables you to use static template files in your application.
- At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.
- This approach makes it easier to design an HTML page.

Template Engine in Express JS

Simple Input

```
//index.pug

doctype html
html
    head
        title A simple pug example
    body
        h1 This page is produced by pug template engine
        p some paragraph here..
```

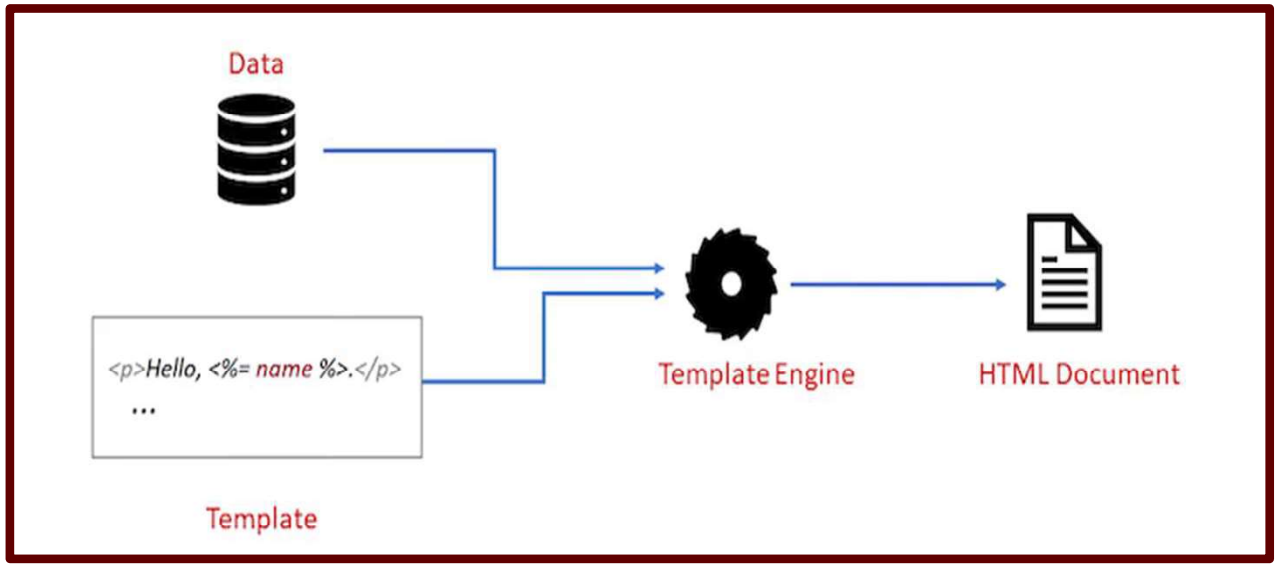
Output Produced by Engine

```
//index.html
<!DOCTYPE html>
<html>
  <head>
    <title>A simple pug example</title>
  </head>
  <body>
    <h1>This page is produced by pug template engine</h1>
    <p>some paragraph here..</p>
  </body>
</html>
```

Template Engine in Express JS

Following is a list of some popular template engines Express.js:

- **EJS Embedded JavaScript Template**
- **Pug (formerly known as jade)**
- mustache
- dust
- atpl
- eco
- ect
- ejs
- haml
- haml-coffee
- handlebars
- hogan



EJS Template Engine in Express JS

- Template engine makes you able to use static template files in your application. To render template files you have to set the following application setting properties:
- **Views:** It specifies a directory where the template files are located.

  For example: app.set('views', './views').

- **view engine:** It specifies the template engine that you use. **For example**, to use the EJS template engine: **app.set('view engine', 'ejs').**

EJS Template Engine in Express JS

- Initialize a new Node project in the folder by running **npm init -y** in the terminal, then to install Express and EJS, run:

npm install express ejs

- create an app.js file and a views folder in the root folder. Inside the views folder, create folder pages.

Note: *Click Here to Access the implementation in Drive*.

EJS Template Engine in Express JS

11/19/22, 10:45 AM app.js

```
1
2  const express = require('express')
3  const app = express()
4  const port = 3000
5  app.set('view engine', 'ejs')
6
7  const user = {
8      firstName: 'Tim',
9      lastName: 'Cook',
10 }
11 app.get('/', (req, res) => {
12     res.render('pages/index', {
13         user: user
14     })
15 })
16 app.listen(port, () => {
17   console.log(`App listening at port ${port}`)
18 })
```

11/19/22, 10:46 AM index.ejs

```
1 <h1>Hi, I am <%= user.firstName  %></h1>
```

Pug Template Engine in Express JS

Install pug

- **Execute the following command to install pug template engine:**

  npm install pug

- Create a file named **index.pug** file inside views folder and write the following pug template in it:

```
//index.pug

doctype html
html
    head
        title A simple pug example
    body
        h1 This page is produced by pug template engine
        p some paragraph here..
```

Pug Template Engine in Express JS

File: server.js

```
var express = require('express');
var app = express();

// Set view engine to Pug
app.set('view engine', 'pug');

// Define a route
app.get('/', function (req, res) {
  res.render('index');
});

// Start the server
var server = app.listen(5000, function () {
  console.log('Node server is running..');
});
```

Template Engine in Express JS

Advantages of Template engine in Node.js

- Improves developer's productivity.
- Improves readability and maintainability.
- Faster performance.
- Maximizes client side processing.
- Single template for multiple pages.
- Templates can be accessed from CDN (Content Delivery Network).

Defining Routes

- Route paths, in combination with a request method, define the endpoints at which requests can be made.

Route paths

- Route paths, in combination with a request method, define the endpoints at which requests can be made. Route paths can be strings, string patterns, or regular expressions.

Defining Routes

```
This route path will match requests to the root route, /.
app.get('/', (req, res) => {
  res.send('root')
})
This route path will match requests to /about.
app.get('/about', (req, res) => {
  res.send('about')
})
This route path will match requests to /random.text.
app.get('/random.text', (req, res) => {
  res.send('random.text')
})
```

Route paths based on string patterns.

This route path will match acd and abcd.

```
app.get('/ab?cd', (req, res) => {
  res.send('ab?cd')
})
```

This route path will match abcd, abbcd, abbbcd, and so on.

```
app.get('/ab+cd', (req, res) => {
  res.send('ab+cd')
})
```

This route path will match abcd, abxcd, abRANDOMcd, ab123cd, and so on.

```
app.get('/ab*cd', (req, res) => {
  res.send('ab*cd')
})
```

Route paths based on regular expressions

This route path will match anything with an “a” in it.

```
app.get(/a/, (req, res) => {
  res.send('/a/')
})
```

This route path will match butterfly and dragonfly, but not butterflyman, dragonflyman, and so on.

```
app.get(/.*fly$/, (req, res) => {
  res.send('/.*fly$/')
})
```

Route parameters

- Route parameters are named URL segments that are used to capture the values specified at their position in the URL.
- The captured values are populated in the req.params object, with the name of the route parameter specified in the path as their respective keys.

Route path: /users/:userId/books/:bookId

Request URL: http://localhost:3000/users/34/books/8989

req.params: { "userId": "34", "bookId": "8989" }

Response Methods

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

Defining Routes-index.html

```
<!DOCTYPE html>
<html>
<body>

<h1>The a element</h1>

<a href="/cbit">Visit cbit.ac.in</a>
<a href="/youtube">Visit YouTube</a>
<a href="/facebook">Visit Facebook</a>

</body>
</html>
```

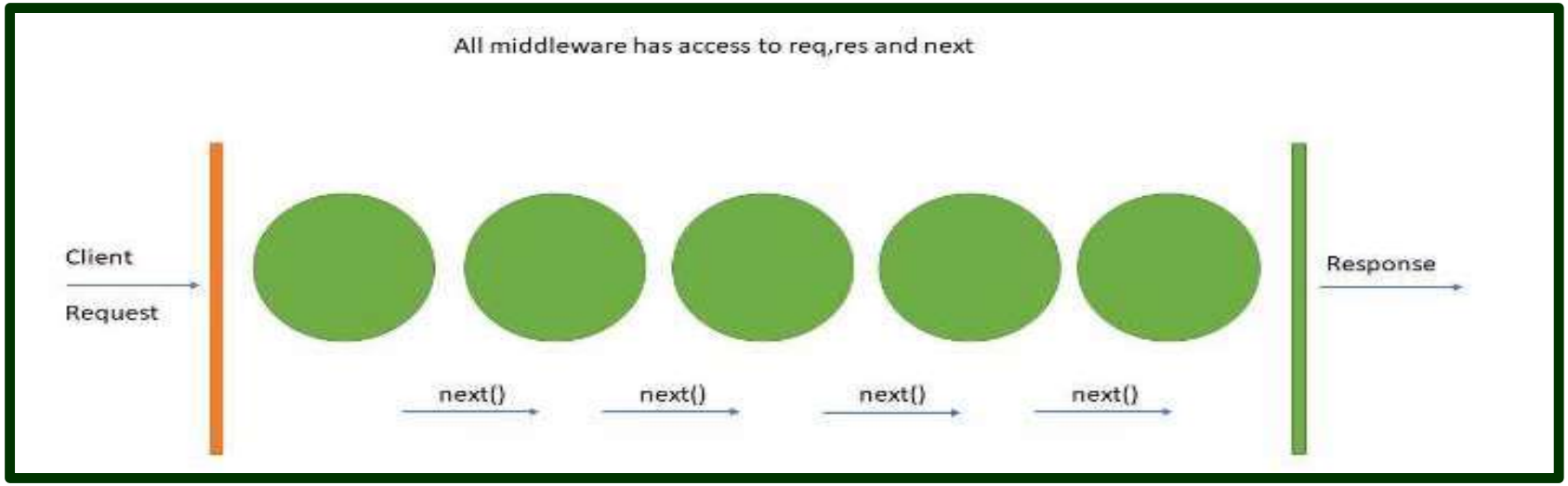
Note: *Click Here to Access the implementation in Drive.*

Defining Routes- index.js

```
// run `node index.js` in the terminal

var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.sendFile(__dirname + '/' + 'index.html');
});
app.get('/cbit', function (req, res) {
  res.redirect('https://www.cbit.ac.in/');
});
// This responds a GET request for abcd, abxcd, ab123cd, and so on
app.get('/facebook', function (req, res) {
  res.redirect('https://www.facebook.com/CBIThyderabad/');
});

app.get('/youtube', function (req, res) {
  res.redirect('https://www.youtube.com/channel/UCUW8oQB8Fl6j-pg2g_sf1tw');
});

app.listen(3000);
```

Middleware in Express

The middleware in node.js is a function that will have all the access for requesting an object, responding to an object, and moving to the next middleware function in the application request-response cycle.



Middleware in Express

Middleware functions can perform the following tasks:

Execute any code.

Make changes to the request and the response objects.

End the request-response cycle.

Call the next middleware in the stack.

Middleware in Express

An Express application can use the following types of middleware.

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

Middleware in Express

The **app.use() function** is used to mount the specified middleware function(s) at the path which is being specified. It is mostly used to set up middleware for your application.

Syntax:

app.use(path, callback)

```
app.use(function(req,res,next)
{
alert("Welcome");
console.log("Requrest Method is",req.method,"and",req.url,"url address page is running");
next();
});
```

Note: ***next(), executes the middleware succeeding the current middleware***

Middleware/Custom Middleware in Express

```
var express = require("express");
var app = express();

app.use(function(req, res, next) {
  console.log("Request Method:", req.method, "URL:", req.url);
  next();
});

app.get('/home', function(req, res) {
  console.log("First Page");
  res.send("Welcome to CBIT First Page");
});

app.get('/exit', function(req, res) {
  console.log("Second Page");
  res.send("Thanks for Your Support");
});

app.use(function(req, res) {
  console.log("The END");
  res.status(404).send("Page Not Found");
});

app.listen(8080, function() {
  console.log('Server is running on http://localhost:8080');
});
```

Note: ***Click Here to Access the implementation in Drive.***