**Node Js**

1. Introduction
   1. About node js - **Done**
   2. Where to use node js - **Done**
   3. Why to use node js - **Done**
   4. let, const, var differences - **Done**
   5. Prerequisites
      1. ES-6, ECMAScript
      2. Arrow functions - **Done**
      3. Call back functions
      4. Destructing
      5. Asynchronous - **Inprogress**
      6. Promises
2. Features of node js - **Done**
3. Node js Workflow - **Done**
4. Node js installation - **Done**
5. Best editor (vs code) installation - **Done**
6. Built-in modules in node js
   1. OS Module - **Done**
      1. To get the OS details
         1. type
         2. version
         3. free space
         4. cpu status
      2. To get the current filename and folder name
   2. PATH Module - **Done**
      1. To get the current location where you are now
      2. To get only the file name
      3. To get the extension of the file
      4. To get the current file folder information
   3. FS Module - **Done**
      1. Reading data from file
      2. Creating a new file
      3. Append content to the file
      4. Replace the content in the file
      5. To rename the file
      6. Checking if a file exists or not
      7. Creating a directory
      8. To delete the file
   4. HTTP Module - **Done**
   5. URL Module - **Done**
   6. Events module -
   7. Upload files -
   8. Emails -
7. Local modules or User defined modules - **Done**
8. npm in node js - **Done**
9. Express Js introduction - **Done**
10. Middlewares - **Done**
11. Creating a web server using express js – **Done**

**Introduction**

1. Node js is not a programing language and not a framework and not a library
2. It is a runtime environment used for executing JavaScript code on server side
3. By using this node js, we can develop / build backend services like APIs
4. We can use this APIs with mobile apps and web applications

**Features of Node Js**

1. Node js is for developing a highly scalable, data intensive and real time apps
2. Special features in node js
   1. To use this node js, no need of strongest environment setup
   2. Node is easy to get started. No need of other software’s to run this node js
   3. No need of bigger kind of environmental setup
   4. It is super-fast and highly scalable service
   5. It has a larger eco system of ­­open-source libs. We can develop highly scalable and less weight applications
   6. Javascript everywhere. We can have javascript, where ever we do programming or developing an application. No need-to-know other programming languages

**Node Js Work flow**

1. Node js is an asynchronous by its nature. Asynchronous means it is a non-blocking kind of operations
2. It will take the multiple requests from the multiple users and at the same time and it can handle the response for multiple / all requests. It can handle multiple requests at the same time and it can give response to multiple requests at the same time
3. It will not wait for other process to complete
4. There is no kind of processing one request after completing the other

**Things to work on**

1. ­­HTTP module
2. File system module
3. URL module
4. Node js NPM
5. Node js Email

**Node Js Installation**

1. From website download node js with LTS
   1. <nodejs.org/en>
2. Run the exe file to complete the installation
3. After installation, we can check the version as below
   1. Open command prompt 🡪 Enter command 🡪 node –version

**Best editor to work with node Js**

1. Visual studio code editor

**Hello world program**

1. By using HTTP module, we can able to create a server and we can render the content whatever we want
2. Create one folder for node js application and open this folder with vs code editor
3. Create server.js file in a folder and add below code

var http= require('http');

http.createServer((req,res)=>{

    console.log('Server is running');

    res.write('Hello world');

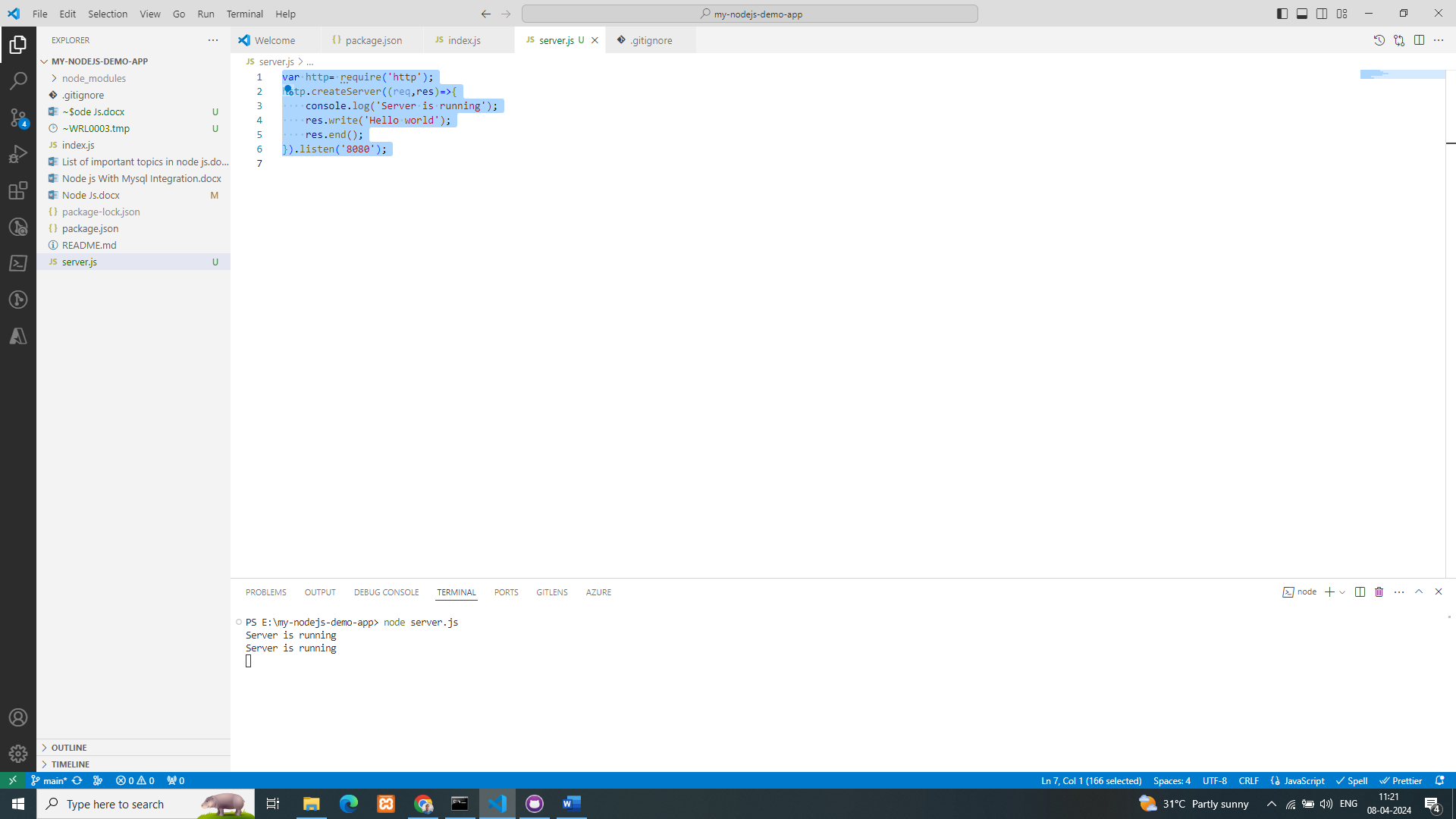
    res.end();

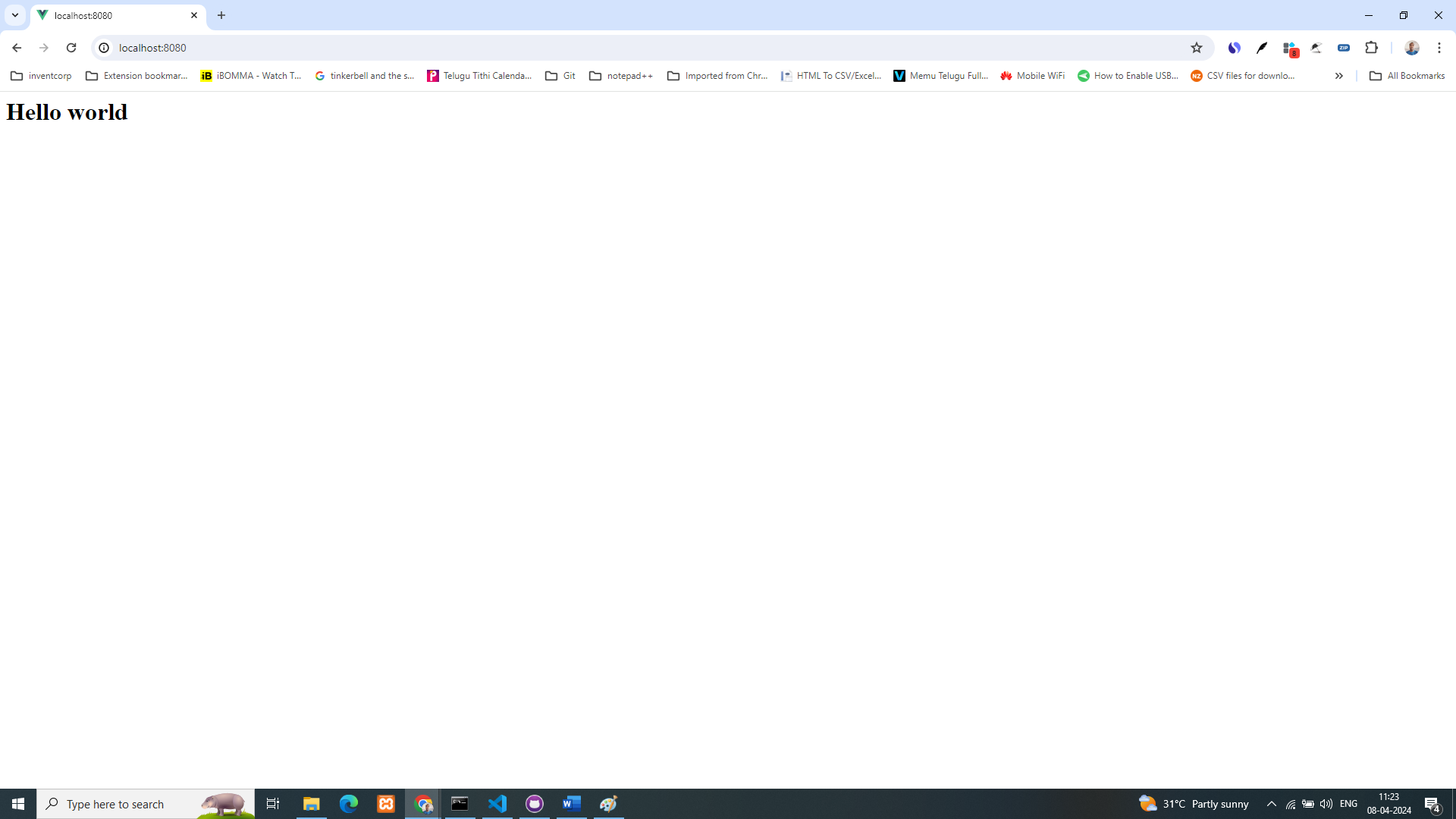
}).listen('8080');

1. Save the file.
2. To start the server, open the terminal.
3. To run the script, run below command in the terminal
   1. node server.js
4. If we add any console.log statements in the script, then we can see the output in terminal. Otherwise, we need to open the browser and enter the below URL
   1. Localhost:8080

**Explore Hello world program**

1. In the above script, we can use **require** to import the http module
2. To run the server in 8080 port, we use **listen** function, We can use any port which is available
3. In the above script, we use **write** function to generate the content for the browser
4. We need to end the response, so we used **end** function at the end of the arrow function
5. We open the url in the browser, then only server will start and run
6. If we are using node modules, then no need to start and stop the server again and again
7. Whatever we pass the parameters in browser url, we can utilize now using req argument in the server-side script
8. Response can be able to write the content to the particular browser





**File system module**

1. To read the file and if you want to display the content in the browser by reading the file at server side, to update the file and to delete the file we use this filesystem module
2. We can shortly call it as fs module and the same use it while writing the script or code
3. To use this module, first we need to import it
4. Script for reading the content from the test.txt file as below

var http= require('http');

var fs = require('fs');

http.createServer((req,res)=>{

    console.log('Server is running');

    // To read the content from the file

    fs.readFile('test.txt',(err,data)=>{

// Here err can be used to handle the errors while reading the file

// Data will store the data after reading the file

        res.write(data);

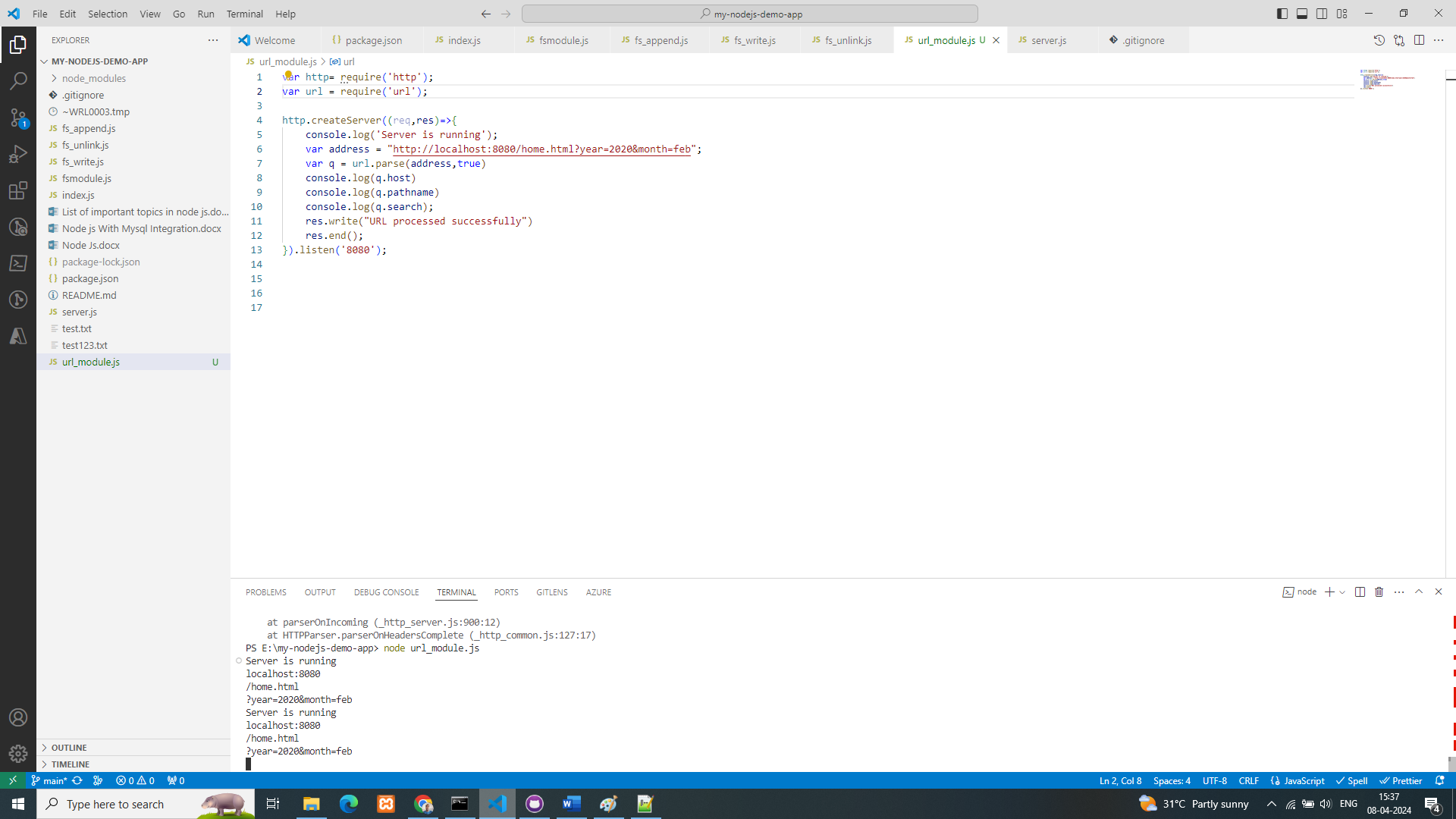
        res.end();

    })

}).listen('8080');

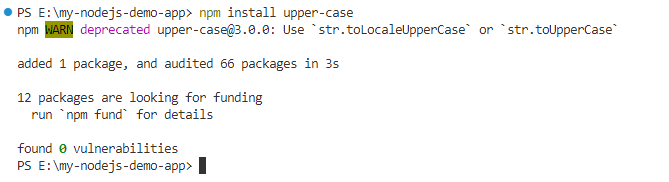
**URL Module**

1. To handle the URL, parameters and values received from the browser at server side
2. To send the values through params from one page to another page. It might be a path params or that might be a query params
3. To use this URL module, we need to import url module first



**Node Js NPM Module**

1. NPM stands for node package manager
2. We can perform multiple kind of activities
3. Using this NPM, we can install libraries and utilize them in our real time applications
4. Example: Install upper-case module
   1. Open the command prompt, change the path if required to where we want to install this module using npm and run below command

  
**Streams**

1. In Node.js, streams are an essential concept for handling data flow efficiently, especially when working with large datasets.
2. Streams provide an interface for reading from or writing to a data source in chunks, rather than loading the entire dataset into memory at once.
3. This allows Node.js applications to process data more efficiently, with lower memory usage and improved performance.
4. Streams can be classified into four main types:
   1. Readable
   2. Writable
   3. Duplex
   4. Transform.

**1. Readable Streams:**

1. Readable streams represent a source from which data can be read.
2. Examples include reading data from a file, HTTP request, or database query result.
3. We can consume data from a readable stream using the read method or by listening to events such as data, end, and error.

**2. Writable Streams:**

1. Writable streams represent a destination to which data can be written.
2. Examples include writing data to a file, making an HTTP request, or inserting data into a database.
3. We can write data to a writable stream using the write method or by piping data from a readable stream.

**3. Duplex Streams:**

1. Duplex streams represent both a readable and writable stream.
2. Examples include TCP sockets, which allow bidirectional communication.
3. Duplex streams can both read from and write to the underlying data source simultaneously.

**4. Transform Streams:**

1. Transform streams are a type of duplex stream that can modify or transform the data passing through them.
2. Examples include compressing or decompressing data, encrypting or decrypting data, and parsing or serializing data.
3. Transform streams allow you to perform data manipulation while the data is being transferred from a readable to a writable stream.

Node.js provides a built-in module called stream to work with streams. We can create custom streams by extending the stream.Readable, stream.Writable, stream.Duplex, or stream.Transform classes. Additionally, Node.js includes several built-in stream classes for common use cases, such as reading from files (fs.createReadStream), writing to files (fs.createWriteStream), and making HTTP requests (http.request).

Using streams in Node.js can lead to more efficient and scalable applications, especially when dealing with large volumes of data. Streams allow you to process data incrementally, reducing memory usage and improving performance. They are an essential tool in the Node.js developer's toolkit for building high-performance, data-intensive applications.

Top of Form

**Modules and requires**

1. In Node.js, **exports** and **require** are key features of the CommonJS module system, which allows you to modularize your code into separate files and import/export functionality between them.
2. **exports**:
   * The **exports** object is used to define what functionality or data from a module should be accessible to other modules.
   * You can assign properties or methods to **exports** to make them available for importing in other modules.
   * Example: **math.js**

exports.add = function(a, b) {

return a + b;

};

exports.subtract = function(a, b) {

return a - b;

};

1. **require**:
   * The **require** function is used to import functionality or data from other modules.
   * When you call **require** with the path to a module, Node.js loads the module and returns the object assigned to its **exports**.
   * Example: **index.js**

const math = require('./math');

console.log(math.add(5, 3)); // Output: 8 console.log(math.subtract(5, 3)); // Output: 2

* In the example above, we have a **math.js** module that exports two functions (**add** and **subtract**) using the **exports** object.
* In **index.js**, we use **require** to import the **math** module, and then we can access its exported functions (**add** and **subtract**) using dot notation.
* We can also assign the entire module to a variable using **require**, and then access its exported properties or methods using dot notation

const math = require('./math');

console.log(math.add(5, 3)); // Output: 8

console.log(math.subtract(5, 3)); // Output: 2

* This flexibility allows you to structure your code into reusable modules and keep your codebase clean and organized.
* We can export any JavaScript value (objects, functions, variables, etc.) from a module using **exports**, and then import it into other modules using **require**.

**Mysql Integration with Node Js**

**Install mysql package with npm as below**

npm install mysql

create a database.js file in the project folder and test the below code. We can give any file name

We have 2 ways to create connection using below methods

1. createConnection
   1. We have to manage the connections from outside means we have to do the connections and we have to close the connection
2. createPool - We can create a poll of a connection not a single connection and we can execute the query parallelly

const mysql = require('mysql');

// Create connection

const connection = mysql.createConnection({

host: 'localhost',

user: 'username',

password: 'password',

database: 'database\_name'

});

// Connect

connection.connect();

// Create table

connection.query('CREATE TABLE IF NOT EXISTS users (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), email VARCHAR(255))', (err, result) => {

if (err) throw err;

console.log('Table created successfully');

});

// Insert data

const userData = { name: 'John', email: 'john@example.com' };

connection.query('INSERT INTO users SET ?', userData, (err, result) => {

if (err) throw err;

console.log('Data inserted successfully');

});

// Update data

connection.query('UPDATE users SET email = ? WHERE name = ?', ['john.doe@example.com', 'John'], (err, result) => {

if (err) throw err;

console.log('Data updated successfully');

});

// Delete table

connection.query('DROP TABLE IF EXISTS users', (err, result) => {

if (err) throw err;

console.log('Table deleted successfully');

});

//Bulk insert

const bulkInsertQuery = `

INSERT INTO users (username, email) VALUES ?

`;

const values = [

['Alice', 'alice@example.com'],

['Bob', 'bob@example.com'],

['Charlie', 'charlie@example.com']

];

connection.query(bulkInsertQuery, [values], (err, results) => {

if (err) throw err;

console.log('Bulk data inserted successfully');

});

//Subqueries

const subQuery = `

SELECT \* FROM users WHERE email IN (SELECT email FROM other\_table WHERE condition)

`;

connection.query(subQuery, (err, results) => {

if (err) throw err;

console.log('Subquery executed:', results);

});

//Inner queries

const innerQuery = `

SELECT \* FROM users WHERE id IN (SELECT MAX(id) FROM users)

`;

connection.query(innerQuery, (err, results) => {

if (err) throw err;

console.log('Inner query executed:', results);

});

// Close connection

connection.end();