**Node.js** is a runtime environment that allows developers to run JavaScript code on the server side, outside of a web browser. It is built on the V8 JavaScript engine, which is the same engine used by Google Chrome to execute JavaScript code. Here are some key features and aspects of Node.js:

1. **Asynchronous and Event-Driven**: Node.js uses a non-blocking, event-driven architecture, making it suitable for handling multiple simultaneous connections efficiently. This is particularly useful for real-time applications like chat servers, online gaming, and collaborative tools.
2. **Single-Threaded**: - Despite being single-threaded, Node.js can handle many connections concurrently thanks to its event loop and callback mechanisms, which help manage I/O operations without blocking the execution of other code.
3. **Package Management**: - Node.js includes npm (Node Package Manager), which is the largest ecosystem of open-source libraries available to developers. npm allows developers to easily share and reuse code, speeding up development.
4. **Cross-platform: -** Node.js is cross-platform, meaning it can run on various operating systems like Windows, macOS, and Linux, making it highly versatile for different deployment environments.
5. **Microservices and APIs**: - Node.js is often used for building RESTful APIs and microservices, due to its lightweight and fast nature. It's well-suited for applications that require fast I/O operations, such as database interactions and web services.
6. **Scalability: -** Node.js applications can be scaled horizontally by running multiple instances of the application across different servers or cores, making it suitable for large-scale applications.
7. **Full-Stack Development: -** Node.js allows for full-stack JavaScript development, where developers can use the same programming language (JavaScript) for both client-side and server-side code, streamlining the development process.
8. **Community and Ecosystem: -** Node.js has a vibrant and active community, with a wealth of tutorials, libraries, and tools available to developers. This makes it easier to find solutions and best practices for various challenges.

**Import and Export in Node.js**: - Importing and exporting files are important parts of any programming language. Importing functions or modules enhances the reusability of code. When the application grows in size, maintaining a single file with all the functions and logic becomes difficult. It also hinders the process of debugging. Therefore, it is good practice to create separate files for specific functions and later import them as per requirement.

Node.js also allows importing and exporting functions and modules. Functions in one module can be imported and called in other modules saving the effort to copy function definitions into the other files. The module can be edited or debugged separately making it easier to add or remove features. We are exporting and Importing JSON array and Function.

**To Export Use**: - module.exports ={studentlist, show}

Var studentlist=

[

//**Key** **Value**

{ “name” : “Darpan”,

“age” : 34,

“class” : “React” },

{ “name” : “Vinod”, **JSON ARRAY**

“age” : 36,

“class” : “Node.js” },

{ “name” : “Naman”,

“age” : 26,

“class” : “JavaScript” }

]

Var show=()=>

{Console.log (“Arrow Function Called”);} **Arrow Function**

**To Import to another file use:** - var stlist=require (‘./file name from where data is to be imported)

Now we can use the data of studentlist with stlist and we can print data using

Console.log(“studentlist data is in stlist”, stlist.studentlist)

***We can do the direct: -***

**To Export Use**: - module.exports =studentlist

Var studentlist=

[

//**Key** **Value**

{ “name” : “Darpan”,

“age” : 34,

“class” : “React” },

{ “name” : “Vinod”, **JSON ARRAY**

“age” : 36,

“class” : “Node.js” },

{ “name” : “Naman”,

“age” : 26,

“class” : “JavaScript” }

]

**To Import to another file use:** - var stlist=require (‘./file name from where data is to be imported)

Now we can use the data of studentlist with stlist and we can print data using

Console.log(“studentlist data is in stlist”, stlist)

**Modules in node.js: -** modules are a way to organize and structure your code by encapsulating related functionality into separate files and directories. This modular approach helps in managing code complexity, reusability, and maintainability. Node.js has a robust module system that includes built-in (core) modules, third-party modules, and user-defined modules. Here’s a detailed look at the types of modules in Node.js:

1. **Core Modules: -**

* fs : Provides an API for interacting with the file system.
* http : Used to create HTTP servers and clients.
* path : Utilities for working with file and directory paths.
* os : Provides operating system-related utility methods and properties.
* url : Utilities for URL resolution and parsing.
* crypto : Provides cryptographic functionality including hashing and encryption.

1. **Third party Module: -**

* express : A web application framework for Node.js
* mongoose : An ODM (Object Data Modeling) library for MongoDB and Node.js.
* lodash : A utility library that provides useful functions for common programming tasks.
* Async : Provides powerful functions for working with asynchronous JavaScript.

**Fs Module: -** Provides an API for interacting with the file system.

* fs.writeFile
* fs.appendFile
* fs.readFile

Var fs = require(‘fs’); First we have to import the function of ‘fs’ module to a variable.

fs.writeFile(‘darpan.txt’, ‘content here’, function(err)

{

if (err) throw err; This fs.writeFile creates a new txt file and content written in it.

console.log( ‘Saved!’ );

});

Var fs = require(‘fs’); First we have to import the function of ‘fs’ module to a variable.

fs.appendFile(‘darpan.txt’, ‘content here’, function(err)

{

if (err) throw err; This fs.appendFile update the file and content written in it.

console.log( ‘Updated!’ );

});

Var fs = require(‘fs’); First we have to import the function of ‘fs’ module to a variable.

fs.readFile(‘mehta.txt’, ‘utf-8’, function(err, data)

{

if (err) throw err; This fs.readFile read the data from a file and copy into data in function

console.log( ‘data’ ); and show in the Vs code terminal.

});

**url Module: -** Utilities for URL resolution and parsing. It split a web address into readable parts: -

var url = require(‘url’) First we have to import the function of ‘url’ module to a variable.

var adr = ‘localhost : 8080/default.htm?year=2017&month=februrary’;

var q = url.parse(adr, true);

console.log(q.host);

console.log(q.pathname);

console.log(q.search);

‘localhost : 8080/default.htm?year=2017&month=februrary’;

q.host q.pathname q.search

var k=q.query;

console.log(k.year);

console.log(k.month);

**HTTP Module: -** The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client. Hyper Text Transfer Protocol (HTTP).

Var HTTP = require(‘http’); First we have to import the function of ‘http’ module to a variable

http.createServer(function(req, res)

{

res.write(‘content here’); This function creates the local host server 8080 with the content

res.end(); we provide in it.

}

).listen (8080).

Now picking up the content written on a txt file and printing on the local server using **fs and http module**.

Var http = require (‘http’);

Var fs = require (‘fs’);

http.createServer(function(req, res)

{

fs.readFile(‘mehta.txt’, ‘utf-8’, function(err, data)

{

if (err) throw err; In this we are picking the content from a txt file named as Mehta.txt using fs.readFile

res.write(data); and printing on the local server using http.createServer.

res.end();

})

}

).listen (8080).

**Third-Party Module: -**

**To install express module we have to install a package name *npm install express.***

express module: -

In Node.js, **.get** is commonly used in the context of the Express.js framework, a popular web application framework for Node.js. The **.get** method in Express is used to define a route that listens for HTTP GET requests on a specified path. Here's an overview of how **.get** is used and its purpose:

var express = require('express')

var fs=require('fs')

var app = express();

app.get('/',(req,res)=>{

res.send("welcome to node js express api ") })

app.get('/getuserlist',(req,res)=>{

fs.readFile('mehta.txt','utf-8', function(err, data)

{ if (err) throw err; res.send(data) }) })

app.get('\*\*',(req,res)=>{ res.send

("you have entered wrong route, please provide right one ") })

app.listen(8080, () =>

{ console.log(`Server running on port 8080`); });

In this reading content form other file and posting on local server.

var express = require('express')

var fs=require('fs')

var app = express();

app.use(express.json()); //to understand the json file receiving from postman

app.get('/',(req,res)=>{

res.send("welcome to node js express api ")

})

app.get('/employeedata',(req,res)=>{

fs.readFile('employee.txt','utf-8', function(err, data) {

if (err) throw err;

res.send(data)

})

})

app.post('/employeedetails',(req,res)=>{

fs.appendFile('employee.txt',JSON.stringify(req.body), function (err)

{

if (err) throw err;

console.log('Saved!');

res.send({

status:200,

message:"data is saved successfully"

})

}

);

})

app.get('\*\*',(req,res)=>{

res.send("you have entered wrong route, please provide right one ")

})

app.listen(8080, () => {

console.log(`Server running on port 8080`);

});

In this we have created a post API which can only work in postman software we cannot use in local server. In this example we have created a post API in which a request is receiving from postman software which is creating a new txt file automatically and through fs the data of new file is printing on the local server.

**Jwt (Json Web Token): -**

JWT (JSON Web Token) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.

Imagine you have a library card that proves you are a member of your local library. When you want to borrow a book, you show this card to the librarian. The card has some information about you and is signed by the library to confirm it’s valid.

In the world of web applications, JWT is like that library card:

**Mongodb**

**Netlify**

**Ejs**

**Socket**