sever.js

const {multiplicationOfTwo, generateRandomNumber } = require('./util');

let sum = (a,b) => a + b

let result = sum(2,3)

console.log(result);

let c = generateRandomNumber();

let d = generateRandomNumber();

console.log(` multiplication of ${c} \* ${d} =`, multiplicationOfTwo(c,d));

util.js

//math.random() returns from 0 to 1

function generateRandomNumber(){

return Math.floor(Math.random() \* 10 + 1);

}

function multiplicationOfTwo(a,b){

return a \* b;

}

//exporting module functionName

module.exports = {

generateRandomNumber,

multiplicationOfTwo

} ;

// module.exports = multiplicationOfTwo;

ES modules

Server.js

import  getPostsLength, { getPosts }  from './util.js';

console.log(getPosts());

console.log(` Posts length : ${getPostsLength()}`);

util.js

const posts =[{

    title1: "hello",

    title2: "whatsup"

}]

const getPosts = () => posts;

const getPostsLength =() => posts.length;

export { getPosts };

export default getPostsLength;

The first one is using **CommonJS modules**, and the second one is using **ES modules**. These are two different module systems in JavaScript, and they have distinct syntax and behaviors.

**1. CommonJS Modules (First Example)**

**Key Characteristics:**

* Uses require() to import modules.
* Uses module.exports to export modules.
* This is the default module system in Node.js (unless "type": "module" is specified in package.json).
* Synchronous loading of modules.

**Example:**

**server.js (CommonJS)**

javascript

Copy

const { multiplicationOfTwo, generateRandomNumber } = require('./util');

let sum = (a, b) => a + b;

let result = sum(2, 3);

console.log(result);

let c = generateRandomNumber();

let d = generateRandomNumber();

console.log(`Multiplication of ${c} \* ${d} =`, multiplicationOfTwo(c, d));

**util.js (CommonJS)**

javascript

Copy

function generateRandomNumber() {

return Math.floor(Math.random() \* 10 + 1);

}

function multiplicationOfTwo(a, b) {

return a \* b;

}

module.exports = {

generateRandomNumber,

multiplicationOfTwo

};

**2. ES Modules (Second Example)**

**Key Characteristics:**

* Uses import and export statements.
* Requires "type": "module" in package.json or .mjs file extension.
* Asynchronous loading of modules.
* Native support in modern JavaScript environments (browsers and Node.js).

**Example:**

**server.js (ES Modules)**

javascript

Copy

import getPostsLength, { getPosts } from './util.js';

console.log(getPosts());

console.log(`Posts length: ${getPostsLength()}`);

**util.js (ES Modules)**

javascript

Copy

const posts = [{

title1: "hello",

title2: "whatsup"

}];

const getPosts = () => posts;

const getPostsLength = () => posts.length;

export { getPosts };

export default getPostsLength;

**Key Differences Between CommonJS and ES Modules**

| **Feature** | **CommonJS Modules** | **ES Modules** |
| --- | --- | --- |
| **Syntax** | require() and module.exports | import and export |
| **Default Behavior** | Default in Node.js | Requires "type": "module" or .mjs |
| **Loading** | Synchronous | Asynchronous |
| **Browser Support** | Not natively supported in browsers | Natively supported in modern browsers |
| **Static Analysis** | Less amenable to static analysis | Better for static analysis (tree-shaking) |
| **Dynamic Imports** | Not natively supported | Supported via import() |

**When to Use Which?**

* **CommonJS**: Use this if you're working in a Node.js environment and don't want to deal with ES module-specific configurations.
* **ES Modules**: Use this if you're working in a modern JavaScript environment (e.g., browsers, Node.js with "type": "module") and want to leverage modern JavaScript features.