NodeJS

LABA

MODULE SYSTEM

import vs require

Solvd
NodeJS LABA

WHAT IS COMMONJS

logger.js

```
function log(message) {
  const timestamp = new Date().toISOString();
  console.log(`${timestamp}: ${message}`);
}

module.exports = log;
```

app.js

```
const log = require("./logger");

log("Starging the application...");
// Additional application logic.
log("Application is running");
// More application logic.
log("Application finished execution");
```

Solvd
NodeJS LABA

WHAT IS COMMONJS

a.js

```
function sayName(name) {
  console.log(`My name is ${name}.`);
};

function sayAge(age) {
  console.log(`I'm ${age} years old.`);
};

module.exports = { sayName, sayAge };
```

app.js

```
const { sayName, sayAge } = require("./a");
// assuming a.js is in the same folder path

console.log(sayName("Alex")); // My name is Alex.

console.log(sayAge(25)); // I'm 25 years old.
```

Solvd
NodeJS LABA

WHAT IS COMMONJS

```
(function(exports, require, module, __filename, __dirname) {
  // Module code actually lives in here
})
```

Solvd
NodeJS LABA

WHAT IS COMMONJS

```
function Module(id = '', parent) {
  this.id = id;
  this.path = path.dirname(id);
  this.exports = {};
  this.parent = parent;
  updateChildren(parent, this, false);
  this.filename = null;
  this.loaded = false;
  this.children = [];
};
```

Solvd
NodeJS LABA

WHAT IS COMMONJS

```
function updateChildren(parent, child, scan) {
  const children = parent && parent.children;
  if (children && !(scan && children.includes(child))) {
    children.push(child);
  }
};
```

Solvd
NodeJS LABA

```
{
  id: '.',
  path: '/Users/ilyakaracun/Documents/tm/ow-gsc-backend',
  filename: '/Users/ilyakaracun/Documents/tm/ow-gsc-backend/test.js',
  loaded: false,
  children: [],
  paths: [
    '/Users/ilyakaracun/Documents/tm/ow-gsc-backend/node_modules',
    '/Users/ilyakaracun/Documents/tm/node_modules',
    '/Users/ilyakaracun/Documents/node_modules',
    '/Users/ilyakaracun/node_modules',
    '/Users/node_modules',
    '/node_modules'
  ]
} [Arguments] {
  '0': {},
  '1': [Function: require] {
    resolve: [Function: resolve] { paths: [Function: paths] },
    main: {
      id: '.',
      path: '/Users/ilyakaracun/Documents/tm/ow-gsc-backend',
      filename: '/Users/ilyakaracun/Documents/tm/ow-gsc-backend/test.js',
      loaded: false,
      children: [],
      paths: [Array]
    },
    extensions: [Object: null prototype] {
      '.js': [Function (anonymous)],
      '.json': [Function (anonymous)],
      '.node': [Function (anonymous)]
    },
    cache: [Object: null prototype] {
      '/Users/ilyakaracun/Documents/tm/ow-gsc-backend/test.js': [Object]
    }
  },
}
```

PROS OF COMMONJS

- Easy to learn and use
- It is widely supported.
- Synchronous module loading ensures that all dependencies are loaded before execution.

CONS OF COMMONJS

- Synchronous module loading can lead to performance issues in larger apps
- Lack of tree-shaking can lead to larger bundle sizes
- Not suitable for client-side development within browsers

WHAT ARE ESMODULES

Modern module system built into the JavaScript language.

Created to solve the problem of managing dependencies in JavaScript projects, both on the client and server-side.

WHAT ARE ESMODULES

```
(node:2844) Warning: To load an ES module, set "type": "module" in the
package.json or use the .mjs extension.(Use `node --trace-warnings ...` to show
where the warning was created). Also, as an aside, we cannot make use of import
statements outside of modules.
```

Solvd NodeJS LABA

WHAT ARE ESMODULES

logger.mjs

```
export function log(message) {
  const timestamp = new Date().toISOString();
  console.log(`${timestamp}: ${message}`);
};
```

app.js

```
import { log } from "./logger.mjs";

log("Starting the applicaiton...");
// Additional application logic.
log("Application is running");
// More application logic.
log("Application finished execution");
```

Solvd NodeJS LABA

WHAT ARE ESMODULES

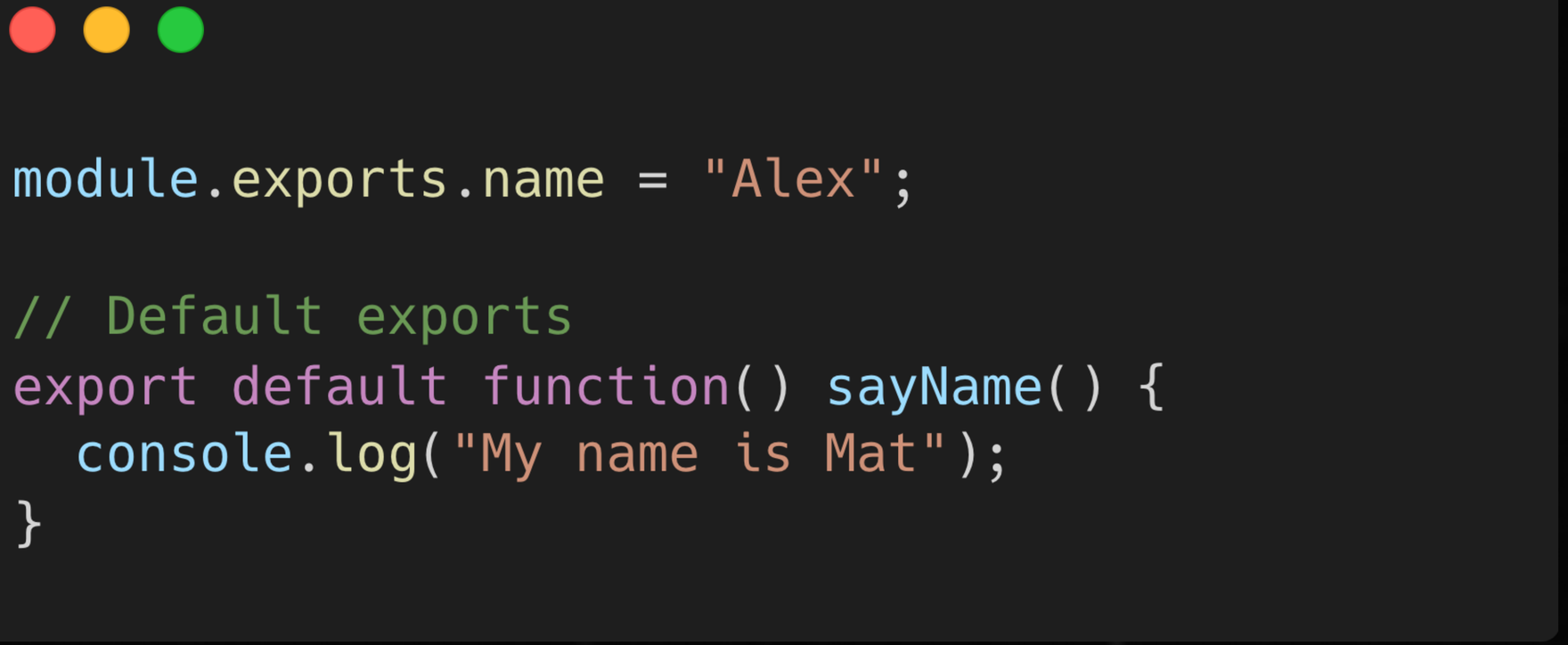
```
import { cat } from "animals";

// Works!
import packageMain from "commonjs-package";

// Errors
import { method } from "commonjs-package";
```

Solvd NodeJS LABA

WHAT ARE ESMODULES



```
module.exports.name = "Alex";

// Default exports
export default function() sayName() {
  console.log("My name is Mat");
}
```

Solvd NodeJS LABA

PACKAGE ENTRY POINTS

The main field in package.json specifies the primary module loaded when a package is imported without a subpath

```
const myPacakge = require("my-package");
```

It is a string pointing to a file relative to the package root

HOW IT WORKS

When Node.js processes `require('my-package')`, it looks for the main field in my-package/package.json. If present, it loads the specified file. If absent, Node.js defaults to index.js in the package root (if it exists).

PURPOSE

Provides a simple way to define the package’s primary entry point, ensuring consumers access the intended module.

LIMITATIONS

The main field only defines a single entry point and doesn’t control access to other files in the package. Users can still import any file by appending a path (e.g., `require(‘my-package/lib/utils.js’)`, which may expose internal implementation details.

HOW IT WORKS

project structure

```
my-package/
├── index.js
├── lib
│   └── utils.js
└── package.json
```

index.js

```js
module.exports = {
  greet: () => "Hello, world!",
};
```

package.json

```json
{
    "name": "my-package",
    "main": "index.js"
}
```

```js
const myPackage = require("my-package");

console.log(myPackage.greet());
// Output: Hello, world!

// However, users can also access
const utils = require("my-package/lib/utils");
```

Solvd
NodeJS LABA

EXPORTS FIELD

The exports field in package.json defines a mapping of entry points for the package, allowing precise control over which modules are accessible.

HOW IT WORKS

When exports is defined, Node.js restricts module access to only the paths listed in the exports object, encapsulating the package. Imports like `require('my-package')` or `require('my-package/subpath')` resolve to the paths defined in exports. Any attempt to access undefined paths results in an error.

PURPOSE

Enhances encapsulation by hiding internal files, improves predictability, and supports conditional exports (e.g., different files for CommonJS vs. ES Modules).

ADVANTAGES

Unlike main, exports prevents direct access to internal files, enforces a clear API, and supports advanced features like conditional exports for different environments

HOW IT WORKS

project structure

```
my-package/
├── index.js
├── lib
│   └── utils.js
└── package.json
```

lib/util.js

```
module.exports = {
  sum: (a, b) => a + b
};
```

package.json

```
{
  "name": "my-package",
  "main": "index.js",
  "exports": {
    ".": "./index.js",
    "./utils": "./lib/utils.js"
  }
}
```

```
const myPackage = require("my-package");
const utils = require("my-package/utils");

console.log(myPackage.greet());
// Output: Hello, world!
console.log(utils.sum(2, 3));
// Output: 5
```

Solvd NodeJS LABA

PROS OF ES MODULES

- ES Modules is a standardized module system built into the JavaScript language.
- Asynchronous module loading can improve performance in large apps.
- The tree-shaking feature can reduce bundle sizes.

CONS OF ES MODULES

- ES Modules is relatively new and not fully supported by older web browsers.
- The syntax for importing and exporting modules can be complex.

NODEJS PATH MODULE

The path module in Node.js is a core module that offers utilities for managing file and directory paths.

It helps handle and transform paths across different operating systems, ensuring platform independence. The module includes methods for joining, resolving, and normalizing paths, among other common file system tasks.

```
const path = require("path");
```

Solvd
NodeJS LABA

WHY USE THE PATH MODULE

Different operating systems (Windows, macOS, Linux) handle file paths differently.

```
// On Windows:
C:\Users\Documents\file.txt
// On Linux/MacOS
/Users/Documents/file.txt
```

Solvd NodeJS LABA

FEATURES OF PATH MODULE

Cross-Platform Path Handling: The path module allows you to work with file paths in a platform-independent manner, ensuring consistent behavior across different operating systems.

Path Resolution and Normalization: It provides methods to resolve absolute paths, normalize paths, and remove redundant path segments, which simplifies file path management.

Path Parsing and Formatting: The module includes functions for extracting and formatting path components such as directory names, file names, and extensions.

Convenient Path Operations : Functions like path.join() and path.resolve() make it easy to concatenate and resolve paths, reducing the likelihood of errors.

path.basename()

The path.basename() method is used to get the filename portion of a path to the file. The trailing directory separators are ignored when using this method. Syntax:

```
const path = require("path");
// path.basename(path, extension);

const path1 = path.basename("/home/user/bash/index.txt");
console.log(path1);
// Output: index.txt

const path2 = path.basename("/home/user/bash/index.txt", ".txt");
console.log(path2);
// Output: index
```

Solvd NodeJS LABA

path.dirname()

```
const path = require("path");

const path1 = path.dirname("/home/user/bash/index.txt");
console.log(path1);
// Output: /home/user/bash

const path2 = path.dirname("readme.md");
console.log(path2);
// Output: .

const path3 = path.dirname("website/post/comments");
console.log(path3);
// Output: website/post
```

path.format()

Parameters: This function accepts single parameter pathObject that contains the details of the path. It has the following parameters:

- dir: It specifies the directory name of the path object.
- root: It specifies the root of the path object.
- base: It specifies the base of the path object.
- name: It specifies the file name of the path object.
- ext: It specifies the file extension of the path object.

path.format()

```
const path = require("path");

const path1 = path.format({
  root: "/ignored/root",
  dir: "/home/user/personal",
  base: "details.txt",
});
console.log("Path 1:", path1);
// Output: Path 1: /home/user/personal/details.txt

const path2 = path.format({
  root: "/",
  base: "game.dat",
  ext: ".noextension"
});
console.log("Path 2:", path2);
// Output: Path 2: /game.dat

const path3 = path.format({
  root: "/images/",
  name: "image",
  ext: ".jpg",
});
console.log("Path 3:", path3);
// Output: Path 3: /images/image.jpg
```

Solvd
NodeJS LABA

path.isAbsolute()

```
const path = require("path");

const path1 = path.isAbsolute("/user/bash/");
console.log(path1);
// Output: true

const path2 = path.isAbsolute("user/bash/readme.md");
console.log(path2);
// Output: false

const path3 = path.isAbsolute("/user/bash/readme.md");
console.log(path3);
// Output: true

const path4 = path.isAbsolute("..");
console.log(path4);
// Output: false
```

path.join()

Parameters:

- **path1, path2, …, pathN: These are the path segments that you want to join together. You can provide multiple path segments, and the function will join them correctly.**

Return Value:

- **It returns a string with the complete normalized path containing all the segments.**

path.join()

```
const path = require("path");

const path1 = path.join("user/admin/files", "index.html");
console.log(path1);
// Output: users/admin/files/index.html

const path2 = path.join("users", "geeks/website", "index.html");
console.log(path2);
// Output: users/geeks/website/index.html

const path3 = path.join("users", "", "", "index.html");
console.log(path3);
// Output: users/index.html
```

Solvd
NodeJS LABA

path.join()

```
const path = require("path");

const path1 = path.join("users", "..", "files", "readme.md");
console.log(path1);
// Output: files/readme.md

const path2 = path.join("users", "..");
console.log(path2);
// Output: .

console.log("Current directory:", __dirname)
const path3 = path.join(__dirname, "..");
console.log(path3);
// Output: Current directory: /Users/ilyakaracun/Documents/tm/ow-gsc-backend
// Output: /Users/ilyakaracun/Documents/tm
```

path.normalize()

```
const path = require("path");

const path1 = path.normalize("/users/admin/.");
console.log(path1);
// Output: \users\admin

const path2 = path.normalize("/users/amin/..");
console.log(path2);
// Output: \users

const path3 = path.normalize("/users/admin/../comments");
console.log(path3);
// Output: \users\admin\comments

const path4 = path.normalize("/users///admin///comments");
console.log(path4);
// Output: \users\admin\comments
```

Solvd
NodeJS LABA

path.parse()

The path.parse() method is used to return an object whose properties represent the given path. This method returns the following properties:

- root (root name)
- dir (directory name)
- base (filename with extension)
- ext (only extension)
- name (only filename)

path.parse()

```
const path = require("path");

const path1 = path.parse("/users/admin/website/index.html");
console.log(path1);

const path2 = path.normalize("website/readme.md");
console.log(path2);
```

```
{
  root: '/',
  dir: '/users/admin/website',
  base: 'index.html',
  ext: '.html',
  name: 'index'
}
{
  root: '',
  dir: 'website',
  base: 'readme.md',
  ext: '.md',
  name: 'readme'
}
```

Solvd NodeJS LABA

SUMMARY

The path module in Node.js is a powerful tool for managing and manipulating file paths in a cross-platform manner.

Providing a variety of methods for joining, resolving, and parsing paths, it ensures that your Node.js applications can handle file operations consistently and efficiently, regardless of the operating system.