Introduction to Node.js

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside a web browser. It's built on Chrome's V8 JavaScript engine and is designed to build scalable network applications.

First Node.js Program

Create a file named app.js:

```
console.log("Hello, Node.js!");
```

open terminal

```
node app.js
```

output

```
Hello, Node.js!
```

Exporting Functions, Variables, and Objects in nodejs

In Node.js, you can export functions, variables, or objects from one module to another using the `module.exports` or exports syntax. This allows you to split your code into multiple files and keep it organized.

Example 1: Exporting a Single Function

```
// file: math.js
// Define a function
function add(a, b) {
  return a + b;
}

// Export the function
module.exports = add;
```

```
// app.js
// Import the function from math.js
const add = require("./math");
```

```
// Use the imported function
console.log(add(2, 3)); // Output: 5
```

Example 2: Exporting multiple functions

```
// file: math.js:

// Define functions
function add(a, b) {
  return a + b;
}

function subtract(a, b) {
  return a - b;
}

// Export multiple functions as an object
module.exports = {
  add,
  subtract,
};
```

```
// file: app.js:

// Import the functions from math.js
const { add, subtract } = require("./math");

// Use the imported functions
console.log(add(2, 3)); // Output: 5
console.log(subtract(5, 3)); // Output: 2
```

Example 3: Exporting Variables and Objects

```
// file: config

// Define variables
const dbHost = "localhost";
const dbUser = "root";
const dbPassword = "";

// Export variables as an object
module.exports = {
  dbHost,
  dbUser,
  dbPassword,
};
```

```
// file: app.js:

// Import the variables from config.js
const config = require("./config");

// Use the imported variables
console.log(config.dbHost); // Output: localhost
console.log(config.dbUser); // Output: root
console.log(config.dbPassword); // Output:
```

Node.js Core Modules

File System

Reading a file:

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

fs.readFile("example.txt", "utf8", (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

Path

Working with file paths:

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

const filePath = path.join(__dirname, "example.txt");
console.log(filePath);
```

HTTP

Creating a simple server:

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

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  res.end("Hello, World!\n");
});
```

```
server.listen(3000, "127.0.0.1", () => {
  console.log("Server running at http://127.0.0.1:3000/");
});
```

---

npm (Node Package Manager)

Initializing a Project

```
npm init
```

Installing a Package

```
npm install express
```

Using a Package

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

app.get("/", (req, res) => {
  res.send("Hello, Express!");
});

app.listen(3000, () => {
  console.log("Server is running on port 3000");
});
```

Interacting with a Database

Using MySQL:

```
// in terminal
// npm install mysql
npm install mysql2
```

```
const mysql = require("mysql2");

const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
```

```
  database: "mydb",
});

connection.connect();

connection.query("SELECT * FROM users", (err, results) => {
  if (err) throw err;
  console.log(results);
});

connection.end();
```

CORS

CORS stands for Cross-Origin Resource Sharing. It is a security feature implemented by web browsers to prevent web pages from making requests to a different domain than the one that served the web page. This helps in preventing various security issues like Cross-Site Request Forgery (CSRF) attacks.

How CORS Works

When a web application running at one origin (e.g., http://example.com) attempts to request resources from a different origin (e.g., http://another-domain.com), the browser sends a preflight request (an OPTIONS request) to the server to check if the CORS protocol is understood and allowed by the server.

The server needs to respond with appropriate CORS headers to indicate whether the request from the other origin is allowed. If the server allows the request, it responds with headers that specify the allowed origins, methods, and headers.

Adding CORS to an Express Application

To enable CORS in an Express application, you can use the cors middleware. This middleware allows you to specify which origins are permitted to access resources on your server.

Step-by-Step Guide to Adding CORS

- Install the cors package: You need to install the cors package using npm:

```
npm install cors
```

- Use the cors middleware in your Express application: Modify your app.js to include and configure the cors middleware.

```
// Use the cors middleware
app.use(cors()); // Enable CORS for all routes
```
