Introduction to Node JS & Node JS Modules

Node.js Basics

1. **What is Node.js and what is it used for?**
   - *Node.js is a runtime environment that allows you to run JavaScript on the server side.*
   - *It is used for building scalable and efficient network applications, particularly web servers and real-time applications.*

2. **Explain the main differences between Node.js and traditional web server environments like Apache or Nginx.**
   - **Concurrency Model:** Node.js uses a non-blocking, event-driven architecture, allowing it to handle many connections simultaneously with a single thread. In contrast, Apache and Nginx use multi-threaded or multi-process models to handle concurrent connections.
   - **Language:** Node.js allows you to write server-side code in JavaScript, whereas Apache and Nginx typically require server-side scripting languages like PHP, Python, or Ruby.
   - **Built-in Capabilities:** Node.js has built-in support for handling HTTP requests and building web servers, whereas Apache and Nginx are dedicated web servers and often rely on external applications or modules for dynamic content.

3. **What is V8 engine and how does Node.js utilize it?**

   The V8 engine is Google's open-source JavaScript engine, which compiles JavaScript directly to native machine code for fast execution. Node.js utilizes the V8 engine to execute JavaScript code on the server side, enabling high performance and efficient resource usage.

4. **Describe the event-driven architecture of Node.js.**

   The event-driven architecture of Node.js is based on an event loop that continuously listens for and processes events. When an event occurs, such as an incoming HTTP request, Node.js uses callbacks or event handlers to manage the event asynchronously. This allows Node.js to handle many operations concurrently without blocking the execution, making it efficient and scalable for I/O-intensive tasks.

5. **What are some common use cases for Node.js?**

   - ***Web Servers:*** *Building fast and scalable web servers.*
   - ***Real-time Applications:*** *Developing chat applications, online gaming, and collaboration tools.*
   - ***APIs:*** *Creating RESTful APIs and microservices.*
   - ***Streaming Applications:*** *Handling real-time data streaming and processing.*
   - ***Single Page Applications (SPAs):*** *Serving SPAs with a smooth, dynamic user experience.*
   - ***IoT Applications:*** *Managing data and communication for Internet of Things devices.*

6. **How does Node.js handle asynchronous operations?**

   Node.js handles asynchronous operations using an event-driven, non-blocking I/O model. It relies on callbacks, promises, and async/await to manage these operations. When an asynchronous operation is initiated, Node.js continues executing the remaining code without waiting for the operation to complete. Once the operation finishes, a callback is triggered or a promise is resolved/rejected, allowing the program to handle the result. This approach enables Node.js to efficiently manage multiple operations concurrently.

7. **What is the purpose of the package.json file in a Node.js project?**

   The purpose of the `package.json` file in a Node.js project is to manage the project's metadata and dependencies. It includes information such as the project name, version, description, author, license, and scripts, as well as a list of dependencies and

devDependencies needed for the project. This file helps in easily sharing, installing, and managing the project's dependencies and configurations.

8. **Explain the role of the Node Package Manager (NPM).**

   The Node Package Manager (NPM) is a tool that manages the installation, updating, and removal of Node.js packages. It allows developers to easily share and reuse code by accessing a vast repository of open-source packages, handling dependencies, and automating various development tasks through scripts defined in the `package.json` file.

9. **What is the node_modules folder and why is it important?**

   The node_modules folder is where Node.js stores all the installed dependencies and their dependencies for a project. It is important because it contains the actual code and files of the libraries and packages specified in the package.json file, enabling the project to function correctly by providing all necessary external modules.

10. **How can you check the version of Node.js and NPM installed on your system?**

    - *To check the version of Node.js by running the command `node -v`*
    - *The version of NPM by running the command `npm -v*

11. **How does Node.js handle concurrency and what are the benefits of this approach?**

    Node.js handles concurrency using a single-threaded, event-driven, non-blocking I/O model. It employs an event loop that allows it to efficiently manage multiple tasks concurrently without creating additional threads or processes for each task. This approach benefits Node.js in several ways:

    - *Node.js can handle a large number of concurrent connections efficiently due to its non-blocking nature, making it suitable for building highly scalable applications.*
    - *By avoiding the overhead of thread creation and context switching, Node.js can execute I/O operations and asynchronous tasks faster, resulting in improved performance.*
    - *Node.js uses fewer resources (CPU and memory) compared to traditional multi-threaded servers, making it more efficient for handling I/O-bound operations.*

(Resource Efficiency)

- *Developers can write cleaner, more maintainable code using asynchronous programming patterns like callbacks, promises, and async/await, without worrying about complex concurrency issues.*

12. How does Node.js handle file I/O? Provide an example of reading a file asynchronously.

Node.js handles file I/O asynchronously using non-blocking operations, which allows it to continue executing other tasks while waiting for file operations to complete.

Example:

```
const fs = require('fs');
fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
    return;
  }
  console.log('File content:', data);
});
```

In this example:

- *fs.readFile is used to asynchronously read the contents of example.txt.*
- *'utf8' specifies the encoding of the file data.*
- *The callback function (err, data) is executed once the file is read. If there's an error (err), it's logged to the console. Otherwise, the file contents (data) are printed.*

13. What are streams in Node.js and how are they useful?

Streams in Node.js are objects that allow you to read or write data sequentially. They provide an abstraction to work with data from a source or to a destination in chunks, rather than loading the entire dataset into memory at once. Streams are useful because:

- ***Memory Efficiency**: They enable processing of large datasets without loading everything into memory, which reduces memory usage.*

- ***Performance****: Streams facilitate faster data processing by handling data in chunks, especially useful for I/O operations like reading from files or network sockets.*
- ***Pipe Operations****: Streams can be easily piped together to create powerful data processing pipelines, simplifying complex data transformations.*
- ***Handling Big Data****: They are ideal for handling data that exceeds available memory, such as log files, large database exports, or real-time data streams.*

Node.js Modules

1. **What are modules in Node.js and why are they important?**

   Modules in Node.js are encapsulated units of functionality that can be reused across different parts of a Node.js application. They allow you to organize your code into separate files or folders, each containing related functions, classes, or variables. Modules are important because:

   - *They help in organizing code into smaller, manageable units, improving readability and maintainability.*
   - *Modules encapsulate related functionality, reducing namespace collisions and promoting cleaner code architecture.*
   - *Modules can be easily reused in different parts of the application or in other projects, promoting code reuse and reducing redundancy.*

2. **How do you create a module in Node.js? Provide a simple example.**

   To create a module in Node.js, typically we have to define the functionalities within a file and use module.exports or exports to expose those functionalities to use in other parts of your application.

   Example:

```
const add = (a, b) => {
   return a + b;
};
const subtract = (a, b) => {
   return a - b;
};
```

```
exports.add = add;
exports.subtract = subtract;
```

3. **Explain the difference between require and import statements in Node.js.**

***require* Statement:**

- *Syntax: CommonJS syntax, e.g., const module = require('module').*
- *Usage: Used to import modules and dependencies in Node.js applications.*
- *Dynamic: require is evaluated at runtime, allowing dynamic module loading.*
- *Default Export: Supports default exports through module.exports.*

***import* Statement:**

- *Syntax: ES Modules (ESM) syntax, e.g., import module from 'module' or import { namedExport } from 'module'.*
- *Usage: Standardized syntax in modern JavaScript for importing modules.*
- *Static: import is statically analyzed during module loading, offering better optimization and tree-shaking capabilities.*
- *Named Exports: Supports named exports and default exports explicitly through export statements.*

4. **What is module.exports object and how is it used?**

In Node.js, module.exports is used to export functions, objects, or classes from a module so they can be used in other files.

Example:

1. ***Exporting a Function:***

```
function add(a, b) {
   return a + b;
}
module.exports = add;
```

2. ***Importing a Function:***

```
const add = require('./add');
console.log(add(2, 3)); // Outputs: 5
```

5. **Describe how you can use the exports shorthand to export module contents.**

The exports shorthand in Node.js is a convenient way to export multiple properties or methods from a module. It is a reference to module.exports.

Example:

```
exports.add = function(a, b) {
return a + b;
};
exports.subtract = function(a, b) {
return a - b;
 };
const math = require('./math');
console.log(math.add(2, 3)); // Outputs: 5
console.log(math.subtract(5, 2)); // Outputs: 3
```

6. **What is the CommonJS module system?**

The CommonJS module system is a standard used in Node.js to structure and organize JavaScript code into modular, reusable components.

Example:

```
function add(a, b) {
   return a + b;
}
function subtract(a, b) {
   return a - b;
}
module.exports = { add, subtract };
const math = require('./math');
console.log(math.add(2, 3)); // Outputs: 5
console.log(math.subtract(5, 2)); // Outputs: 3
```

7. How can you import a module installed via NPM in your Node.js application?

To import a module installed via NPM in your Node.js application:

I. *Install the Module:*

npm install <module_name>

Ex: npm install express

II. *Require the Module:*

const moduleName = require('<module_name>');

Ex: // app.js

```
const express = require('express');
const app = express();
app.get('/', (req, res) => {
   res.send('Hello, world!');
});
app.listen(3000, () => {
   console.log('Server is running on port 3000');
});
```

8. Explain how the path module works in Node.js. Provide an example of using it.

The Path module provides a way of working with directories and file paths.

Example:

```
var path = require('path');
var filename = path.basename('/Users/Refsnes/demo_path.js');
console.log(filename);
```

9. How do you handle circular dependencies in Node.js modules?

- *Circular dependencies in Node.js modules occur when two or more modules require each other directly or indirectly, forming a loop in their dependency graph.*
- *To handle circular dependencies we can,*
  - ✓ *Emit events between modules to communicate without direct dependencies.*
  - ✓ *Refactor the code to avoid circular dependencies.*
  - ✓ *Pass dependencies as parameters to functions rather than requiring*

modules directly.

10. **What is a built-in module in Node.js? Name a few and explain their purposes.**

Built-in modules in Node.js are pre-installed with the runtime environment, offering essential functionalities without needing additional packages.

Examples:

- *fs (File System): Manages file operations.*
- *http (HTTP): Creates HTTP servers and clients.*
- *path (Path): Manipulates file and directory paths.*
- *crypto (Crypto): Provides cryptographic operations.*
- *os (Operating System): Retrieves operating system information.*

11. **What is the difference between relative and absolute module paths in Node.js?**

In Node.js, relative and absolute module paths refer to how you specify the location of modules when using require() to import them into the code.

- *Relative paths are based on the current file's location, while absolute paths specify an exact file system path.*
- *Relative paths are commonly used within the same project, while absolute paths might be necessary when referencing modules outside the project or in specific locations.*
- *Relative paths are more flexible and easier to manage within projects, whereas absolute paths provide explicit control over module locations.*

12. **What is a module wrapper function in Node.js?**

In Node.js, the module wrapper function is an immediately invoked function expression (IIFE) that encapsulates every module's code. It provides:

- *Encapsulation: Ensures module code doesn't pollute the global scope.*
- *Context: Supplies variables (exports, require, module, __filename, __dirname) for module-specific operations.*
- *Module Loading: Facilitates how Node.js manages module scope and dependencies internally.*

13. **Describe the buffer module and its use in Node.js.**

The buffer module in Node.js is used to handle binary data, which consists of raw information like files, images, or network data. It provides a way to store and manipulate this data directly in memory. Here's why it's useful:

- *Storing Data: Buffers can hold binary data like numbers or characters.*
- *Manipulating Data: You can change or read data directly in a buffer.*
- *Applications: Used for tasks like reading files, sending data over networks, or encryption.*

Starting an HTTP Server in Node.js

1. **How do you create a simple HTTP server in Node.js? Provide a code example.**

I. *Import http: We import the built-in http module to create the server.*

II. *Create server: The http.createServer function creates a server object. The provided callback function handles incoming requests.*

III. *Handle request: The callback function takes two arguments: req (request object) and res (response object).*

IV. *Send response: We directly use res.end to send a simple "Hello!" message as the response with the server ending the connection after sending.*

V. *Start server: The server.listen method starts the server, listening on port 3000 in this case.*

```
const http = require('http');
http.createServer((req, res) => {
  res.end('Hello!');
}).listen(3000);
```

2. **Explain the purpose of the http module in Node.js.**

In Node.js, the http module is a built-in library that acts as a foundation for building web applications. It offers functionalities for both creating servers and making requests as a client.

3. **What method do you use to start the HTTP server and make it listen on a specific port?**

   To start an HTTP server and make it listen on a specific port in Node.js, use the listen method of the HTTP server object (server). Provide the desired port number and optionally the hostname

   Example:

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, World!\n');
});
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

4. **How can you send a response to the client in an HTTP server created with Node.js?**

   To send a response to the client in an HTTP server created with Node.js:

   I. *Use res.writeHead(statusCode, headers) to set the HTTP status code and headers.*

   II. *Use res.end(data) to send data as the response body to the client.*

   Example:

```
res.writeHead(200, { 'Content-Type': 'text/plain' });
res.end('Hello, World!\n');
```

5. **Explain the request and response objects in the context of an HTTP server.**

   I. *Request Object (req):*

   - *Represents the client's HTTP request.*
   - *Contains details like URL, method (req.method), headers (req.headers), and data (req.body for POST requests).*

II. *Response Object (res):*

- *Used to send an HTTP response back to the client.*
- *Allows setting status codes (res.statusCode), headers (res.setHeader()), and sending data (res.end()).*

6. How do you handle different HTTP methods (GET, POST, etc.) in a Node.js HTTP server?

By using the built in HTTP module in node.js, we can handle different methods like GET, POST,etc.. by;

I. *Import the http module:*

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

II. *Create the server*

```
const server = http.createServer((req, res) => {
 res.setHeader('Content-Type', 'application/json');

 const url = new URL(req.url, `http://${req.headers.host}`);

 switch (req.method) {
  case 'GET':
   handleGetRequest(req, res, url);
   break;
  case 'POST':
   handlePostRequest(req, res, url);
   break;
  default:
   res.statusCode = 405;
   res.end(JSON.stringify({ message: 'Method Not Allowed' }));
 }
});
```

III. *Handle GET request*

```
function handleGetRequest(req, res, url) {
```

```
  if (url.pathname === '/some-endpoint') {
    res.statusCode = 200;
    res.end(JSON.stringify({ message: 'GET request received' }));
  } else {
    res.statusCode = 404;
    res.end(JSON.stringify({ message: 'Not Found' }));
  }
}
```

IV. *Handle POST request*

```
function handleGetRequest(req, res, url) {
  if (url.pathname === '/some-endpoint') {
    res.statusCode = 200;
    res.end(JSON.stringify({ message: 'GET request received' }));
  } else {
    res.statusCode = 404;
    res.end(JSON.stringify({ message: 'Not Found' }));
  }
}
```

V. *Start the server*

```
const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

7. What is middleware in the context of a Node.js HTTP server?

Functions that process requests and responses as they pass through the server are middlewares. Middleware can be used for tasks such as logging, authentication, parsing request bodies, and handling errors.

8. How can you serve static files using an HTTP server in Node.js?

By serving the static files using an HTTP server in Node.js by using the express.static

middleware in the Express framework.

Example:

```
const express = require('express');
const app = express();
app.use(express.static('public'));
app.listen(3000);
```

9. **Explain how to handle errors in an HTTP server created with Node.js.**

   Handle errors in a Node.js HTTP server by checking for errors in callbacks and using try-catch blocks for synchronous code, or middleware for Express.

   Example:

```
const express = require('express');
const app = express();
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
app.listen(3000);
```

10. **How can you implement routing in a Node.js HTTP server without using external libraries?**

    Implement routing in a Node.js HTTP server by checking req.url and req.method within the server callback function.

    Example:

```
const http = require('http');
const server = http.createServer((req, res) => {
  if (req.url === '/' && req.method === 'GET') {
    res.end('Home Page');
  } else if (req.url === '/about' && req.method === 'GET') {
    res.end('About Page');
  } else {
```

```
    res.statusCode = 404;
    res.end('Not Found');
  }
});
server.listen(3000);
```

Submitted by: UKI STU 879 (Safra Siyam)