NODE JS
OVERVIEW

https://nodejs.org/en

HAJAR BENDHIBA

CONTENT

- Introduction to Node.js
- REPL Terminal & Node Package Manager (NPM)
- Callback Functions in Node.js
- Synchronous vs Asynchronous File System in Node.js
- Event Loop & Event Emitter
- Node.js Global Objects
- Web Server and Client in Node.js
- Creating a Web Server with Node.js
- Creating a Web Client with Node.js

INTRODUCTION TO NODE.JS

Node.js is a JavaScript is an open-source runtime that allows developers to run JavaScript on the server side, enabling the creation of scalable and efficient web applications.

- **Node.js Installation**
  1. Download Node.js from the official site.
  2. Install and verify using:

     node -v

     npm -v

- **Node.js = Runtime + JavaScript library**

- **Creating Node.js Application**
  1. Hello World Example:

```
const http = require('http');
http.createServer((req, res) => {
    res.write('Hello, World!');
    res.end();
}).listen(8080);
```

  2. Run Command:

```
node app.js
```

REPL TERMINAL & NODE PACKAGE MANAGER (NPM)

- **REPL Terminal**

1. REPL stands for Read-Eval-Print Loop, an interactive environment for running commands where:

Read: Reads input and stores it in memory.
Eval: Evaluates the input.
Print: Displays the result.
Loop: Repeats until the user exits.

2. To start the REPL, run node

- **Installing Modules using NPM**

npm --version
npm install npm -g
npm install <module-name>
npm install express
node_modules
npm install express -g

- **Updating a Module**

npm update express

- **Search a Module**

npm search express

CALLBACK FUNCTIONS IN NODE.JS

- **What are Callbacks?**

  An asynchronous functions passed as arguments for async operations.

- **Blocking and Non-Blocking Code**

  Node.js uses synchronous (blocking) methods and also supports asynchronous (non-blocking) methods.

- **Blocking:**

  The program waits for each operation to finish before moving on, leading to sequential execution.

- **Code:**

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');
console.log(data.toString());
console.log("Program Ended");
```

- **Non-Blocking:**

  The program doesn't wait for operations to finish, allowing it to handle multiple tasks concurrently.

- **Code:**

```
var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
 if (err) return console.error(err);
 console.log(data.toString());
}); console.log("Program Ended");
```

SYNCHRONOUS VS ASYNCHRONOUS FILE SYSTEM IN NODE.JS

Node.js's File System (fs) module supports file operations with synchronous and asynchronous methods, favoring asynchronous for non-blocking execution.

- **Node.js File I/O methods:**





	Read a File	Write a File	Append to a File	Delete a File	Rename a File
SYNC	fs.readFileSync	fs.writeFileSync	fs.appendFileSync	fs.unlinkSync	fs.renameSync
ASYNC	fs.readFile	fs.writeFile	fs.appendFile	fs.unlink	fs.rename

SYNCHRONOUS VS ASYNCHRONOUS FILE SYSTEM IN NODE.JS

Opening a File

SYNC fs.openSync

ASYNC fs.open

Getting File Info

fs.statSync

fs.stat

Closing a File

fs.closeSync

fs.close

Truncating a File

fs.ftruncateSync

fs.ftruncate

Managing Directories

Create	ASYNC	fs.mkdir	SYNC	fs.mkdirSync
Read	ASYNC	fs.readdir	SYNC	fs.readdirSync
Delete	ASYNC	fs.rmdir	SYNC	fs.rmdirSync

EVENT LOOP & EVENT EMITTER

Node.js is single-threaded, event-driven, and handles concurrency by triggering events for asynchronous tasks, enabling efficient request management. The Event Loop, with EventEmitter and events module, links events to handlers, boosting performance and scalability.

- **How the Event Loop Works**

```
var events = require('events');
var fs = require('fs');  // Import events module
var eventEmitter = new events.EventEmitter();    // Create an eventEmitter object
var connectHandler = function connected() {
  console.log('Connection successful.');    // Create an event handler
eventEmitter.emit('data_received');};   // Fire the 'data_received' event
eventEmitter.on('connection', connectHandler);   // Bind the 'connection' event with the handler
eventEmitter.on('data_received', function() {
  console.log('Data received successfully.');});   // Bind the 'data_received' event with a function
eventEmitter.emit('connection');   // Fire the 'connection' event
fs.readFile('input.txt', function (err, data) {
  if (err) {
    console.log(err.stack);
    return;
  }  console.log(data.toString());
});   console.log("Program Ended.");  // Demonstrate fs.readFile() for asynchronous file reading
```

NODE.JS GLOBAL OBJECTS

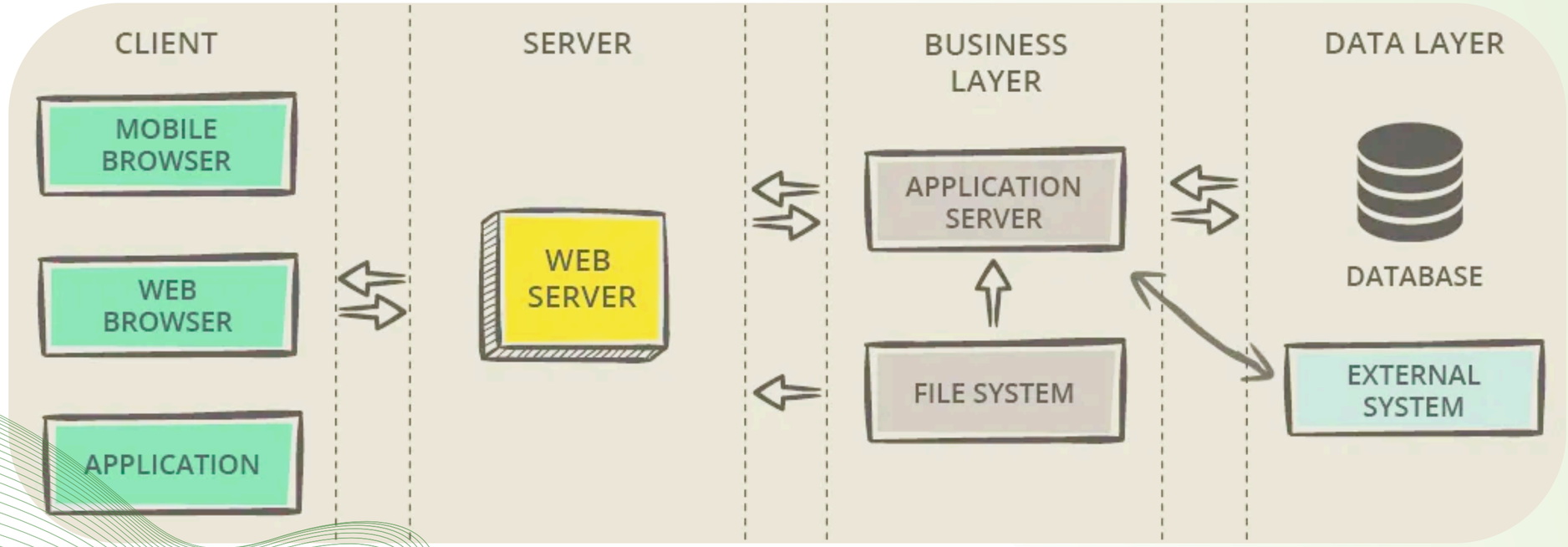
Global Availability: Node.js provides global objects (e.g., modules, functions, strings, objects) available across all modules without importing them.

- **Key Globals**
  - __filename: Gives the full path of the current file being executed. /path/to/main.js.
  - __dirname: Provides the directory name of the current file. /path/to.
  - setTimeout(cb, ms): Runs a callback after a specified delay (in milliseconds).
  - clearTimeout(t): Cancels a timer set by setTimeout.
  - setInterval(cb, ms): Repeats a callback at regular intervals (in milliseconds).
  - clearInterval(t): Stops a timer set by setInterval.

WEB SERVER AND CLIENT IN NODE.JS

A Web Server processes HTTP requests from clients (like browsers) and returns web pages, including HTML, images, and scripts. It may also handle server-side scripts by passing requests to an application server, which retrieves data and sends the response back.

- **Web Application Architecture**



CREATING A WEB SERVER WITH NODE.JS

Use Node.js's http module to create a server that:

- Reads client requests (e.g., /index.html).
- Sends requested content or handles errors (e.g., 404 for missing files).
    - **Example Server code:**

```
var http = require('http');
var fs = require('fs');
var url = require('url');
http.createServer((request, response) => {
   var pathname = url.parse(request.url).pathname;
   fs.readFile(pathname.substr(1), (err, data) => {
      if (err) response.writeHead(404);
      else response.writeHead(200).write(data);
      response.end();
   });
}).listen(8081);
```

CREATING A WEB CLIENT WITH NODE.JS

Use Node.js's http module to create a client that:

- Makes requests to the server.
- Logs the server's response.

- **Example Client code:**

```
var http = require('http');
var options = { host: 'localhost', port: 8081, path: '/index.html' };
http.request(options, (response) => {
   var body = '  ' ;
   response.on('data', (data) => body += data);
   response.on('end', () => console.log(body));
}).end();
```

THANK YOU

HAJAR BENDHIBA


https://www.linkedin.com/in/hajar-bendhiba/


hbendhiba46@gmail.com


+212 7 62 75 23 37


https://github.com/HajarBENDHIBA