## Page:1

# NODE.JS(BASIC)[Lecture-2]

# Interaction with system:

### There are two methods by which we can interact with the system/machine/computer:

A Graphical User Interface (GUI) is a user interface that allows users to interact with electronic devices through graphical icons and visual indicators, as opposed to text-based interfaces, typed command labels, or text navigation. GUIs are designed to be intuitive and user-friendly, enabling users to operate software and hardware systems with minimal training.

## Key Components of a GUI:

1. **Windows:** Rectangular areas that display information and allow interaction. They can be resized, moved, and closed.
2. **Icons:** Small graphical representations of programs, files, functions, or commands. Clicking on an icon typically performs an action or opens a program.
3. **Menus:** Lists of options or commands presented to the user. Menus can be drop-down, pop-up, or fixed in place.
4. **Buttons:** Interactive elements that perform a specific action when clicked.
5. **Toolbars:** Collections of buttons or menus offering quick access to frequently used commands or tools.
6. **Text Fields:** Areas where users can input text data.
7. **Sliders and Scrollbars:** Controls for adjusting values or navigating through content.
8. **Dialog Boxes:** Windows that prompt the user to enter information or make decisions.

## Real-Life Examples of GUIs:

1. **Operating Systems:**
   * **Windows OS:** Microsoft Windows features a GUI with a desktop, start menu, taskbar, icons, and windows for different applications. Users can easily navigate, open programs, and manage files.
   * **macOS:** Apple's macOS uses a GUI with a dock, menu bar, Finder windows, and application icons. It emphasizes ease of use and a visually appealing interface.
2. **Web Browsers:**
   * **Google Chrome:** A web browser with a GUI that includes tabs, a URL address bar, navigation buttons (back, forward, refresh), and menus for settings and options.

## Page:2

* + Users interact with the internet by clicking links, entering search queries, and managing tabs.

1. **Mobile Operating Systems:**
   * **iOS:** Apple's mobile operating system uses a touch-based GUI with icons for apps, a home screen, gestures for navigation, and interactive elements like buttons and sliders.
   * **Android:** Google's mobile OS features a GUI with home screens, app icons, notification bars, and various touch-based interactions.

GUI ⇒ Graphical User Interface 💡 This is the pretty common way that we use to interact with the system, that includes taking the help of graphics to demonstrate and visualise all the actions that we are doing

A Command-Line Interface (CLI) is a text-based user interface used to interact with software and operating systems. Unlike GUIs, which use graphical elements, CLIs rely on text input from the user and text output from the system.

## Key Components of a CLI:

1. **Command Prompt:** The location in the terminal where users enter commands. It often includes a symbol (like $ or >), indicating the system is ready to receive input.
2. **Commands:** Text instructions entered by the user to perform specific tasks. Commands are often followed by options or arguments to modify their behavior.
3. **Shell:** The program that interprets and executes commands entered by the user. Common shells include Bash (Bourne Again Shell) on Unix/Linux systems and Command Prompt or PowerShell on Windows.
4. **Scripts:** Files containing a series of commands that can be executed sequentially by the shell.

CLI ⇒ Command Line Interface

💡 This kind of interaction mainly involves writing specific commands to

perform any kind of operation in the system.

## Page:3

# Common CLI commands

Please explore the following CLI commands, they are really fun:

1. ls - Directory listing

2. is - Formatted listingwith hidden files

3. ls-lt – sorting the formatted listing by time modification

4. cd dir – change directory

5. cd – change to home directory

6. pwd – show current working directory

7. mkdir dir – creating a directory to dir

8. cat>file – places the standard input into the file

9. more file – output the contents of the file

10. head file – output the first 10 lines of the file

11. tail file – output the last 10 lines of the file

12. tail -f file – output the content of file as it grows, starting with the last 10 lines

13. touch file – create or update file

14. rm file – deleting the file

15. rm -r dir – deleting the directory

16. rm -f file – force to remove the file

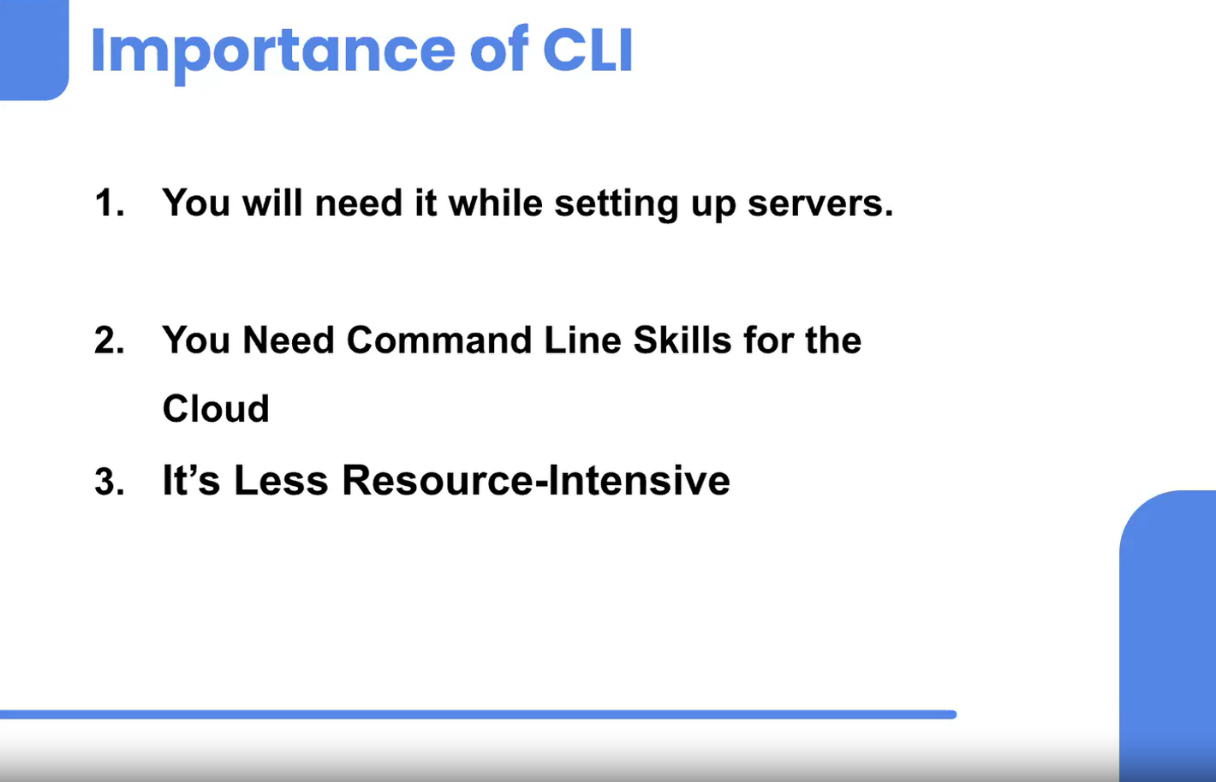
17. rm -rf dir – force to remove the directory dir

18. cp file1 file2 – copy the content of file 1 to file 2

19. cp-r dir1 dir2 – copy dir 1 to dir2 ,create dir 2 if not present

20.mv file1 file2 – rename or move file 1 to file 2,if file 2 is an existing directory

## Page:4



The Command-Line Interface (CLI) holds significant importance in various aspects of computing and development. Here are the key points highlighting its importance:

### 1. Efficiency and Speed

- \*\*Quick Execution:\*\* Commands can be executed rapidly without the need for navigating through menus and windows.

- \*\*Batch Processing:\*\* Allows for the execution of multiple commands and tasks simultaneously through scripting, enhancing productivity.

### 2. Precision and Control

- \*\*Exact Commands:\*\* Users can specify exact commands and parameters, providing granular control over tasks.

- \*\*Advanced Options:\*\* Many CLI tools offer advanced options and settings that are not available in GUI counterparts.

### 3. Resource Efficiency

## Page:5

- \*\*Lightweight:\*\* CLIs consume minimal system resources compared to GUIs, making them ideal for low-resource environments.

- \*\*Remote Access:\*\* Enables efficient management of remote systems over low-bandwidth connections, as it transmits only text.

Node.js supports two module systems: CommonJS (CJS) and ECMAScript Modules (ESM). These systems define how modules are imported and exported in JavaScript. Here’s a detailed comparison and discussion of why you might choose one over the other.

# CJS

Cjs is short for commandjs.here is what it looks like

//importing

const dosomething = requires(‘./dosomething.js’)

//exporting

module.exports = function dosomething(n){

//do something

}

# ESM

Import React from rect

### 1>CommonJS (CJS)

CommonJS is the original module system in Node.js. It is synchronous and works well for server-side applications where modules are typically loaded from disk.

\*\*Example:\*\*

\*\*math.js\*\*

## Page:6

javascriptCopy code

function add(a, b) {

return a + b;

}

function subtract(a, b) {

return a - b;

}

module.exports = { add, subtract };

**app.js**

javascriptCopy code

const math = require('./math');

console.log(math.add(2, 3)); // 5

console.log(math.subtract(5, 3)); // 2

### 2>ECMAScript Modules (ESM)

ECMAScript Modules (ESM) are the standardized module system in JavaScript, introduced in ES6 (ECMAScript 2015). They provide a more modern, flexible, and asynchronous module system.

\*\*Example:\*\*

\*\*math.mjs\*\*

javascriptCopy code

export function add(a, b) {

return a + b;

}

## Page:7

export function subtract(a, b) {

return a - b;

}

**app.mjs**

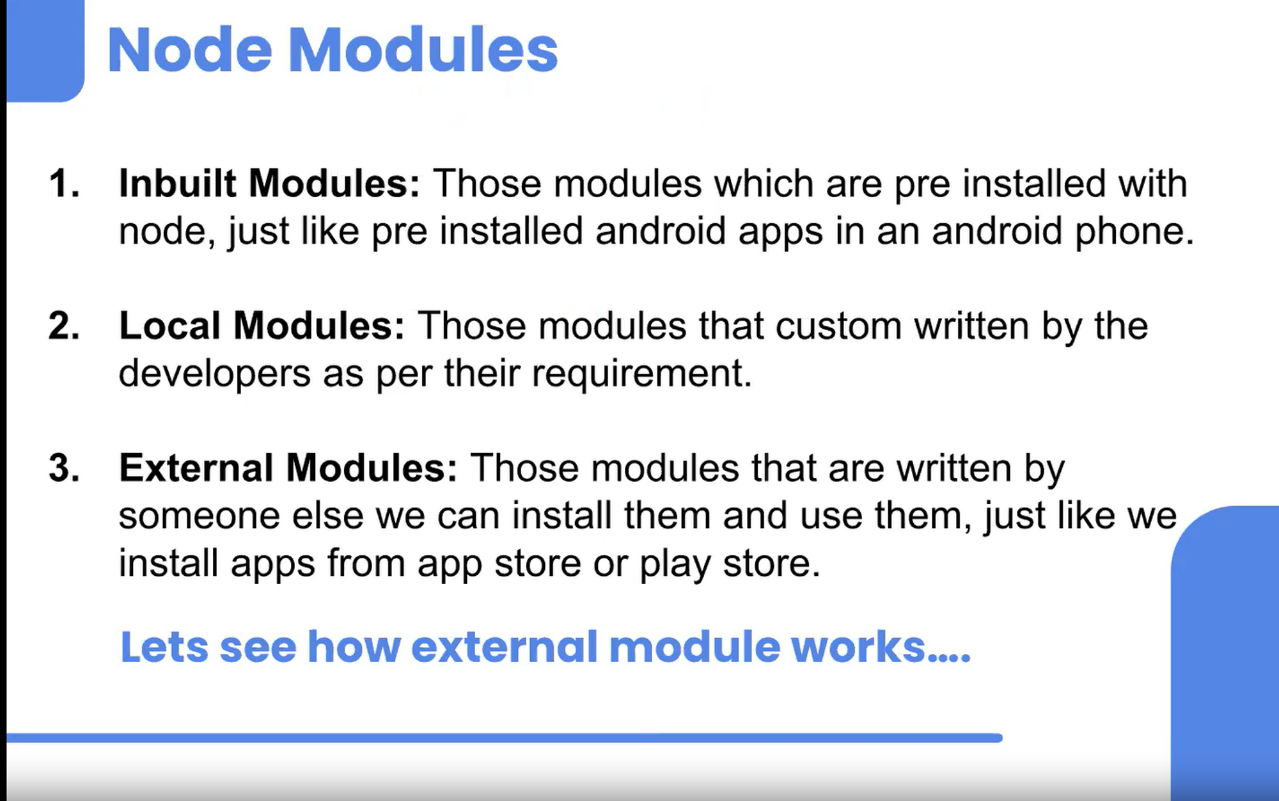
javascriptCopy code

import { add, subtract } from './math.mjs';

console.log(add(2, 3)); // 5

console.log(subtract(5, 3)); // 2

In Node.js, a module is a reusable piece of code that can be included and used in other parts of an application. Node.js uses the CommonJS module system to manage these modules, allowing developers to organize their code into smaller, manageable pieces.



### \*\*Modules Are Three Types:\*\*

\*\*Local Modules:\*\*

## Page:8

Local modules are modules that you create in your own project. They are specific to your project and are not part of the Node.js core or external modules.

Example of a local module:

// utils.js (local module file)

const add = (a, b) => {

return a + b;

};

const subtract = (a, b) => {

return a - b;

};

module.exports = {

add,

subtract,

};

**External Node Modules:**

These are also called as third party libraries, as they are written by someone else and we can directly use them in our project. Install is-even package using npm . Play around with it as we did in the session to understand the concept. You can install any other package as well and plat around with it.

**Core Modules/Inbuild Modules**

* These are modules that come pre-installed with Node.js, and you can use them directly without needing to install anything separately.
* Examples include modules like **fs** (File System), **http** (HTTP server and client), **path** (File and Directory Paths), and **util** (Utilities for debugging and formatting)

**Example:**

Start with the project 💡 Do all the following things using CLI only

## Page:9

**Step-by-Step Guide**

**1. Create a Node.js Project**

Initialize a new Node.js project by creating a directory and running npm init to set up a package.json file.

bashCopy code

mkdir my-node-project

cd my-node-project

npm init -y

**2. Create index.js File**

Create an index.js file which will be the main entry point for your application.

bashCopy code

touch index.js

**3. Create text.txt File**

Create a text.txt file and write some content inside it.

bashCopy code

echo "Hello, this is a sample text file." > text.txt

**4. Read the Content of text.txt Using fs Module**

**Asynchronous Reading:**

The readFile method reads the file asynchronously. The callback function is called once the file read operation is complete.

javascriptCopy code

const fs = require("fs");

fs.readFile("./text.txt", { encoding: "utf-8" }, (err, data) => {

if (err) {

console.log("Cannot read the file");

console.log(err);

} else {

console.log(data);

## Page:10

}

});

console.log("Bye Guys!!");

**Synchronous Reading:**

The readFileSync method reads the file synchronously, blocking the execution until the file reading is complete.

javascriptCopy code

const fs = require("fs");

const data = fs.readFileSync("./text.txt", { encoding: "utf-8" });

console.log(data);

console.log("Bye Guys!!");

**5. Write to a File Using fs Module**

**Asynchronous Writing:**

The writeFile method writes to a file asynchronously. If the file doesn't exist, it will be created.

javascriptCopy code const fs = require("fs"); fs.writeFile("./log.txt", "This is my first time writing to the file", (err) => { if (err) { console.log("Cannot write to the file"); console.log(err); } else { console.log("Data has been written to the file"); } });

**Synchronous Writing:**

The writeFileSync method writes to a file synchronously. Execution is blocked until the writing operation is complete.

javascriptCopy code

const fs = require("fs");

fs.writeFileSync("./log.txt", "This is my second time writing to the file");

console.log("Data has been written to the file");

### 6. Append to a File Using `fs` Module

\*\*Asynchronous Appending:\*\*

## Page:11

The `appendFile` method appends data to a file asynchronously. If the file doesn't exist, it will be created.

javascriptCopy code

const fs = require("fs");

fs.appendFile("./log.txt", "\nThis is my third time writing to the file\n", (err) => {

if (err) {

console.log("Cannot append to the file");

console.log(err);

} else {

console.log("Data has been appended to the file");

}

});

**Synchronous Appending:**

The appendFileSync method appends data to a file synchronously. Execution is blocked until the appending operation is complete.

javascriptCopy code const fs = require("fs"); fs.appendFileSync("./log.txt", "\nThis is my fourth time writing to the file\n"); console.log("Data has been appended to the file");

**Additional Resources**

1. **File System Documentation:** [Node.js File System](https://nodejs.org/dist/latest-v18.x/docs/api/fs.html)
2. **Learning Node.js:** [Node.js Documentation](https://nodejs.dev/en/learn/)

By following this guide, you will have a clear understanding of how to create a Node.js project and use the fs module to read from and write to files in both asynchronous and synchronous manners.