R.M.K GROUP OF ENGINEERING INSTITUTIONS

R.M.K
GROUP OF
INSTITUTIONS

R.M.K
GROUP OF INSTITUTIONS

R.M.K
GROUP OF
INSTITUTIONS

R.M.K
GROUP OF
INSTITUTIONS

Please read this disclaimer before proceeding:

22CS402

Web Development Frameworks

Department : CSE & AI&DS

Batch / Year : 2022 - 2026 / II

Created by : Dr. K Manikannan & Mr. S Vijayakumar, Mr.L.Maria Michael Visuwasam

Date : 03.01.2024

R.M.K GROUP OF INSTITUTIONS

1. Contents

S. No.	Contents
1	Contents
2	Course Objectives
3	Prerequisites
4	Syllabus
5	Course Outcomes
6	CO-PO Mapping
7	Lecture Plan
8	Activity Based Learning
9	Lecture Notes
10	Assignments
11	Part- A Questions & Answers
12	Part-B Questions
13	Supportive Online Courses
14	Real Time Applications
15	Content beyond the Syllabus
16	Assessment Schedule
17	Prescribed Text books & Reference Books
18	Mini Project Suggestions

R.M.K GROUP OF INSTITUTIONS

Content – Unit 3

S.No.	Contents
1	Introduction to Node JS
2	Setting Up Node JS
3	Node JS Modules
4	Finding and Loading common js and json modules using require
5	Finding and Loading common js and json modules using require
6	Hybrid common JS/Node.js/ES6 module scenarios
7	Hybrid common JS/Node.js/ES6 module scenarios
8	Npm the node package management system
9	Npm the node package management system

R.M.K GROUP OF INSTITUTIONS

2. Course Objectives

The Course will enable learners to:

- Simplify website development using Spring Boot as server-side technologies.
- Build single page applications using REACT as a reusable UI component technology as client-side technology.
- Assemble REACT as a front end technology and Node js as a server side technology to develop enterprise applications
- Develop a scalable and responsive web application
- Develop an industry ready application web enterprise feature

3. Prerequisites

22CS402
Web Development Frameworks

22CS301
Advanced Java Programming

22CS202
Java Programming

22CS101 Problem Solving Using C++
22CS102 Software Development Practices

R.M.K GROUP OF INSTITUTIONS

4. Syllabus

22CS301	Web Development Frameworks	L	T	P	C
		3	0	2	4

OBJECTIVES:

The Course will enable learners to:

- Simplify website development using Spring boot as server-side technologies.
- Build single page applications using REACT as a reusable UI component technology as client-side technology.
- Assemble REACT as a front end technology and Node js as a server side technology to develop enterprise applications
- Develop a scalable and responsive web application
- Develop an industry ready application web enterprise feature

UNIT I	SPRING BOOT AND STRUTS	9 +6

Spring Boot: Introducing Spring Boot, getting started with spring Boot, Common Spring Boot task-Managing configuration, creating custom properties, executing code on Spring Boot application startup, Database access with Spring data, Securing Spring Boot application.

List of Exercise/Experiments:

1. Use Spring Boot to build a Web Application
2. Create REST Service for an Education Site

UNIT II	JAVA REACT	9 +6

React: Introduction to React, Pure React- The Virtual DOM, React Elements, React with JSX, Props, State, and the Component Tree, Enhancing Components- Flux.

List of Exercise/Experiments:

1. Build Search filter in React
2. Display a list in React
3. Create Simple Login form in React

UNIT III	Node JS	9 +6

Node JS: Introduction to Node JS, Setting up Node.js, Node.js Modules- Finding and loading CommonJS and JSON modules using require, Hybrid CommonJS/Node.js/ES6 module scenarios, npm - the Node.js package management system.

List of Exercise/Experiments:

1. Write a node.js program for making external http calls
2. Write a program in node.js to parse the given url.

4. Syllabus Contd...

UNIT IV	WEB FRAMEWORK (ANGULAR) – I	9+6
Introduction- Angular First App, Angular UI with Bootstrap CSS Authentication, Authentication Service, Unsubscribe, Logout and Route Guard Cleanup, Customer Service ,Http Service, Token Interceptor, Multi Provider, Compile-time Configuration, Runtime Configuration, Error Handling. **List of Exercise/Experiments:** 1. Create a Dropdown using Angular UI bootstrap 2. Modify existing components and generating new components using Angular		
UNIT V	**WEB FRAMEWORK (ANGULAR) – II**	**9+6**
Dependancy injection in Angular,Reactive programming in Angular, Laying out pages with Flex Layout, Implementing component communications, Change detection and component lifecycle. **List of Exercise/Experiments:** 1. Launching your app with Angular root module		

4. Syllabus Contd...

OUTCOMES:

Upon completion of the course, the students will be able to:

CO1: Write Web API/RESTful API application programming interface to communicate with Spring boot as a serverside technology.

CO2: Build single page applications using REACT as a reusable UI component technology as client side technology

CO3: Build applications using Node Js as server side technologies

CO4: Able to develop a web application using latest Angular Framework

CO5: Apply various Angular features including directives, components, and services.

TEXT BOOK:

1. Somnath Musib, Spring Boot in Practice, Manning publication, June 2022 (https://www.manning.com/books/spring-boot-in-practice)
2. Alex Banks, Eve Porcello , "Learning React", May 2017, O'Reilly Media, Inc. ISBN: 9781491954621. (https://www.oreilly.com/library/view/learning- react/9781491954614/)
3. David Herron ,"Node.js Web Development - Fourth Edition",2018, Packt Publishing,ISBN: 9781788626859
4. Sukesh Marla, "A Journey to Angular Development Paperback ", BPB Publications.(https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r)
5. Yakov Fain Anton Moiseev, "Angular Development with TypeScript", 2nd Edition.(https://www.manning.com/books/angular-development-with-typescript-Second Edition.

REFERENCES:

1. Sue Spielman, The Struts Framework 1: A Practical guide for Java Programmers||, 1st Edition. Elsevier 2002

LIST OF EQUIPMENTS:

VSCode, Angular JS, React JS, Node JS, Ruby, Django

5. Course Outcomes

Upon completion of the course, the students will be able to:

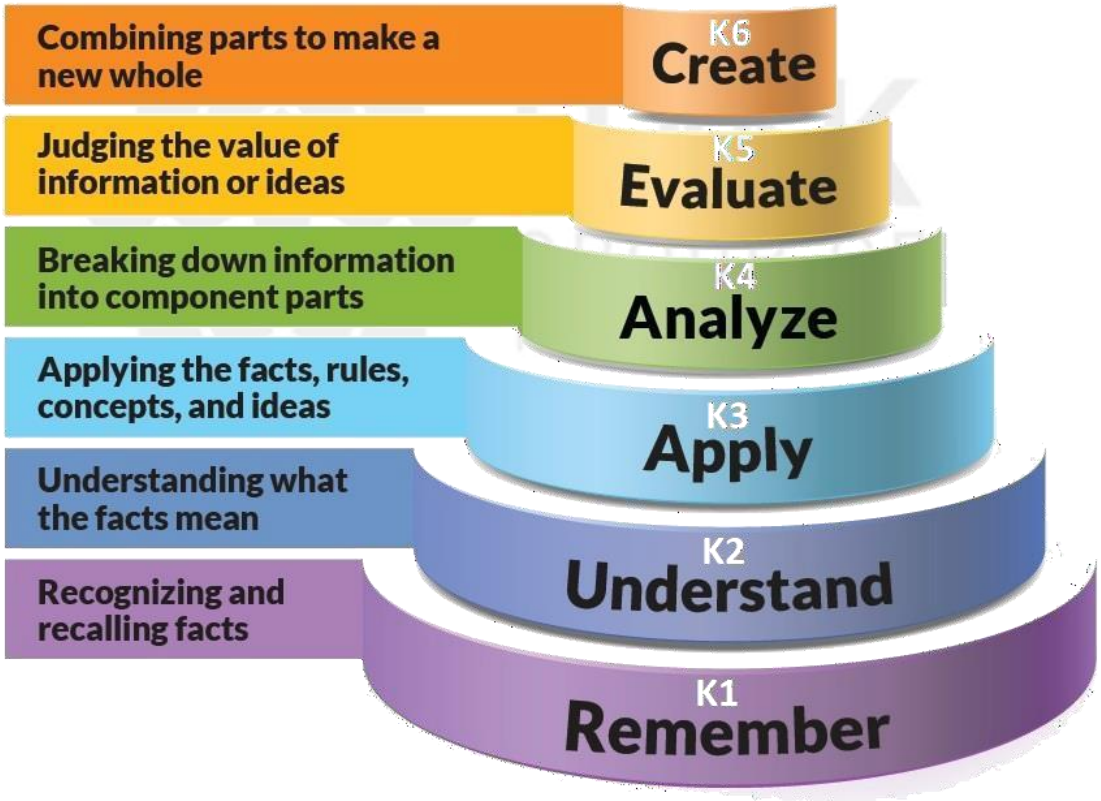
CO1: Write Web API/RESTful API application programming interface to communicate with Spring boot as a serverside technology.

CO2: Build single page applications using REACT as a reusable UI component technology as client side technology

CO3: Build applications using Node Js as server side technologies

CO4: Able to develop a web application using latest Angular Framework

CO5: Apply various Angular features including directives, components, and services.



6. CO - PO Mapping

	POs and PSOs														
COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	2	3	2	2	2	3	2	1	1	3	2	2	3	2	2
CO2	2	3	2	2	2	3	1			3	2	2	3	2	2
CO3	2	3	2	2	2	1	2	1	1	3	2	2	3	2	2
CO4	2	3	2	2	1	3	2	1	1	3	2	2	3	2	2
CO5	2	3	2	2	2	3	2	1	1	3	2	2	3	2	2

7. Lecture Plan - Unit I

S. No.	Topic	No. of Periods	Proposed Date	Actual Lecture Date	Pertaining CO	Taxonomy Level	Mode of Delivery
1	Introduction to Node JS	1			CO3	K2	Chalk & Talk
2	Setting Up Node JS	1			CO3	K3	Chalk & Talk
3	Node JS Modules	1			CO3	K2	Chalk & Talk
4	Finding and Loading common js and json modules using require	1			CO3	K2	Chalk & Talk
5	Finding and Loading common js and json modules using require	1			CO3	K2	Chalk & Talk
6	Hybrid common JS/Node.js/ES6 module scenarios	1			CO3	K2	Chalk & Talk
7	Hybrid common JS/Node.js/ES6 module scenarios	1			CO3	K3	Chalk & Talk
8	NPM-the node package management system	1			CO3	K2	Chalk & Talk
9	NPM-the node package management system	1			CO3	K2	Chalk & Talk

8. Activity Based Learning

Learning Method	Activity
Learn by Solving Problems	Tutorial Sessions available in iamneo Portal
Learn by Questioning	Quiz / MCQ Using RMK Nextgen App and iamneo Portal
Learn by doing Hands-on	Practice available in iamneo Portal

iamneo RMK Nextgen

9. Lecture Notes

Node JS: Introduction to Node JS, Setting up Node.js, Node.js Modules- Finding and loading CommonJS and JSON modules using require, Hybrid CommonJS/Node.js/ES6 module scenarios, npm - the Node.js package management system.

9.LECTURE NOTES

Introduction to Node.js

What is Node.js

Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications.

Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

The definition given by its official documentation is as follows:

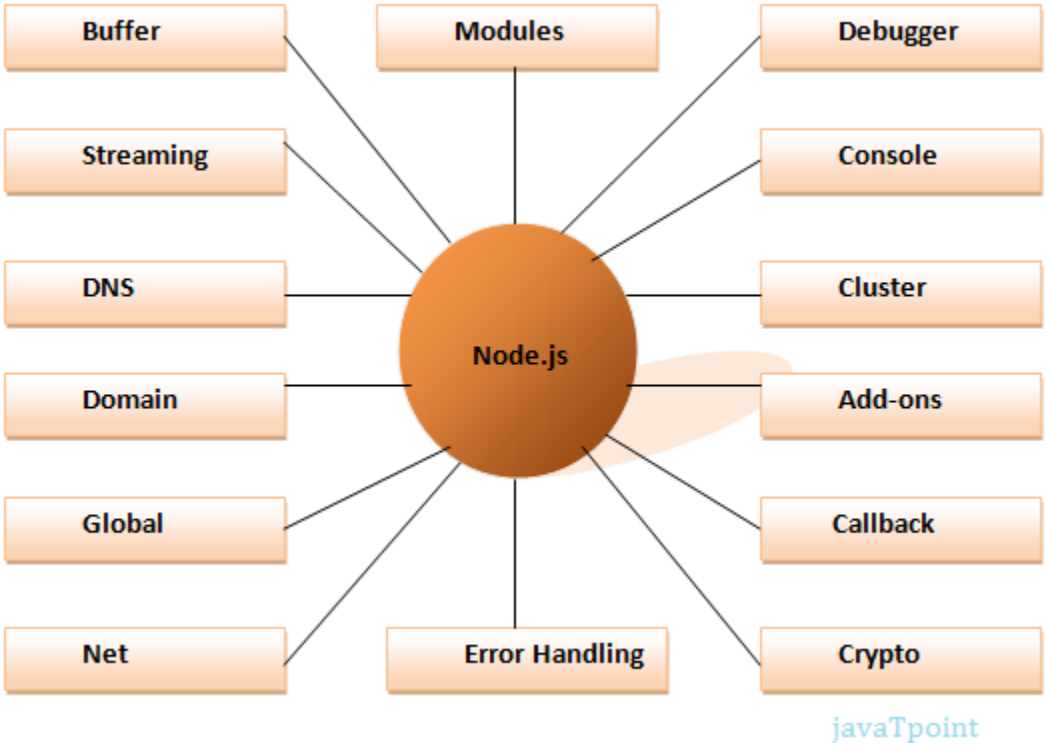
?Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.?

Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.

1. Node.js = Runtime Environment + JavaScript Library

Different parts of Node.js

The following diagram specifies some important parts of Node.js:



Features of Node.js

Following is a list of some important features of Node.js that makes it the first choice of software architects.

1. **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.

2. **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
3. **Single threaded:** Node.js follows a single threaded model with event looping.
4. **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
5. **No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
6. **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License:** Node.js is released under the MIT license.

SETTING UP NODE.JS

Download Node.js archive

Download latest version of Node.js installable archive file from Node.js Downloads. At the time of writing this tutorial, following are the versions available on different OS.

OS	Archive name
Windows	node-v6.3.1-x64.msi
Linux	node-v6.3.1-linux-x86.tar.gz
Mac	node-v6.3.1-darwin-x86.tar.gz
SunOS	node-v6.3.1-sunos-x86.tar.gz

Installation on UNIX/Linux/Mac OS X, and SunOS

Based on your OS architecture, download and extract the archive node-v6.3.1-**osname**.tar.gz into /tmp, and then finally move extracted files into /usr/local/nodejs directory. For example:

```
$ cd /tmp
$ wget http://nodejs.org/dist/v6.3.1/node-v6.3.1-linux-x64.tar.gz
$ tar xvfz node-v6.3.1-linux-x64.tar.gz
$ mkdir -p /usr/local/nodejs
$ mv node-v6.3.1-linux-x64/* /usr/local/nodejs
```

Add /usr/local/nodejs/bin to the PATH environment variable.

OS	Output
Linux	export PATH=$PATH:/usr/local/nodejs/bin

Mac	export PATH=$PATH:/usr/local/nodejs/bin
FreeBSD	export PATH=$PATH:/usr/local/nodejs/bin

Installation on Windows

Use the MSI file and follow the prompts to install the Node.js. By default, the installer uses the Node.js distribution in C:\Program Files\nodejs. The installer should set the C:\Program Files\nodejs\bin directory in window's PATH environment variable. Restart any open command prompts for the change to take effect.

Verify installation: Executing a File

Create a js file named **main.js** on your machine (Windows or Linux) having the following code.

```
/* Hello, World! program in node.js */
console.log("Hello, World!")
```

Now execute main.js file using Node.js interpreter to see the result –

```
$ node main.js
```

If everything is fine with your installation, this should produce the following result –

Hello, World!

Node.js Modules

In Node.js, **Modules** are the blocks of encapsulated code that communicate with an external application on the basis of their related functionality. Modules can be a single file or a collection of multiple files/folders. The reason programmers are heavily reliant on modules is because of their reusability as well as the ability to break down a complex piece of code into manageable chunks.

Modules are of three types:

- Core Modules
- local Modules
- Third-party Modules

Core Modules: Node.js has many built-in modules that are part of the platform and come with Node.js installation. These modules can be loaded into the program by using the **required** function.

Syntax:

```
const module = require('module_name');
```

The require() function will return a JavaScript type depending on what the particular module returns. The following example demonstrates how to use the Node.js http module to create a web server.

javascript

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

http.createServer(function (req, res) {

    res.writeHead(200, { 'Content-Type': 'text/html' });

    res.write('Welcome to this page!');

    res.end();

}).listen(3000);
```

In the above example, the require() function returns an object because the Http module returns its functionality as an object. The function http.createServer() method will be executed when someone tries to access the computer on port 3000. The res.writeHead() method is the status code where 200 means it is OK, while the second argument is an object containing the response headers. The following list contains some of the important core modules in Node.js:

Core Modules	**Description**
http	creates an HTTP server in Node.js.

Core Modules	Description
assert	set of assertion functions useful for testing.
fs	used to handle file system.
path	includes methods to deal with file paths.
process	provides information and control about the current Node.js process.
os	provides information about the operating system.
querystring	utility used for parsing and formatting URL query strings.
url	module provides utilities for URL resolution and parsing.

Local Modules: Unlike built-in and external modules, local modules are created locally in your Node.js application. Let's create a simple calculating module that calculates various operations. Create a calc.js file that has the following code:

Filename: calc.js

javascript

```
exports.add = function (x, y) {
    return x + y;
};
 exports.sub = function (x, y) {
    return x - y;
};
 exports.mult = function (x, y) {
    return x * y;
};
 exports.div = function (x, y) {
    return x / y;
};
```

Since this file provides attributes to the outer world via exports, another file can

use its exported functionality using the require() function.

Filename: index.js

javascript

```
const calculator = require('./calc');

let x = 50, y = 10;

console.log("Addition of 50 and 10 is "

            + calculator.add(x, y));

console.log("Subtraction of 50 and 10 is "

            + calculator.sub(x, y));

console.log("Multiplication of 50 and 10 is "

            + calculator.mult(x, y));

console.log("Division of 50 and 10 is "

            + calculator.div(x, y));
```

Step to run this program: Run the **index.js** file using the following command:

node index.js

Output:

Addition of 50 and 10 is 60

Subtraction of 50 and 10 is 40

Multiplication of 50 and 10 is 500

Division of 50 and 10 is 5

Note: This module also hides functionality that is not needed outside of the module.

Third-party modules: Third-party modules are modules that are available online using the Node Package Manager(NPM). These modules can be installed in the project folder or globally. Some of the popular third-party modules are Mongoose, express, angular, and React.

Example:

- npm install express
- npm install mongoose
- npm install -g @angular/cli

Node.js Local Module

Node.js comes with different predefined modules (e.g. http, fs, path, etc.) that we use and scale our project. We can define modules locally as Local Module. It consists of different functions declared inside a JavaScript object and we reuse them according to the requirement. We can also package it and distribute it using

NPM.

Defining local module: Local module must be written in a separate JavaScript file. In the separate file, we can declare a JavaScript object with different properties and methods.

Step 1: Create a local module with the filename Welcome.js

- javascript

```
const welcome = {

    sayHello: function () {

        console.log("Hello GeekforGeeks user");

    },

    currTime: new Date().toLocaleDateString(),

    companyName: "GeekforGeeks"

}

 module.exports = welcome
```

Explanation: Here, we declared an object 'welcome' with a function sayHello and

two variables currTime and companyName. We use the module.export to make the object available globally.

Part 2: In this part, use the above module in the app.js file.

- javascript

```
const local = require("./Welcome.js");

local.sayHello();

console.log(local.currTime);

console.log(local.companyName);
```

Explanation: Here, we import our local module 'sayHello' in a variable 'local' and consume the function and variables of the created modules.

Output:

Hello GeekforGeeks user

12/6/2019

GeekforGeeks

Node.js Assert module

Assert module in Node.js provides a bunch of facilities that are useful for the

assertion of the function. The assert module provides a set of assertion functions for verifying invariants. If the condition is true it will output nothing else an assertion error is given by the console.

Install the assert module using the following command:

npm install assert

Note: Installation is an optional step as it is inbuilt Node.js module.

Importing module:

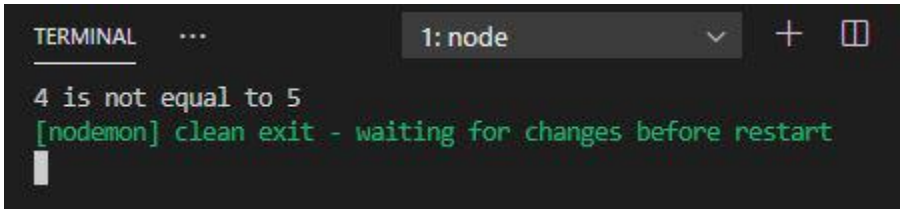
```
const assert = require("assert");
```

Example 1:

```
console.clear()

const assert = require('assert');

  let x = 4;

let y = 5;

  try {

      // Checking condition
```

```
    assert(x == y);

}

catch {

      // Error output

    console.log(

        `${x} is not equal to ${y}`);

}
```

Output:



Example 2:

```
console.clear()
```

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

  let x = 4;

let y = 5;

  assert(x > y);
```

Note: In this example, no try-catch is given so an **assertion error** of the kind given below will be the output.

Output:

```
TERMINAL ···                    1: node

    ^

AssertionError [ERR_ASSERTION]: The expression evaluated to a falsy value
:

  assert(x>y)

    at Object.<anonymous> (E:\web stuff\gfg\Nodejs Assert\script.js:5:1)
    at Module._compile (internal/modules/cjs/loader.js:936:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:947:
10)
    at Module.load (internal/modules/cjs/loader.js:790:32)
    at Function.Module._load (internal/modules/cjs/loader.js:703:12)
    at Function.Module.runMain (internal/modules/cjs/loader.js:999:10)
    at internal/main/run_main_module.js:17:11 {
  generatedMessage: true,
  code: 'ERR_ASSERTION',
  actual: false,
  expected: true,
  operator: '=='
}
```

Node.js Require vs Import

Require in Node.js

As discussed above that Node.js follows the CommonJS module system, and the built-in require function is the easiest way to include modules in this system. When we call require in Node.js then the require function first reads a JavaScript file, executes that file, and then the require function proceeds to return the exports object.

When require function is invoked Node goes through the following sequence of steps:

- **Resolving:**
  In this step Node gets the absolute path of the module. It follows the following steps to resolve the path:
  - For const test = require('example') syntax:
    - It first looks for core modules with that name.
    - If no such core module is present then it looks for a file with the name node_modules/ of the current folder and still if no module is found then it will look for it in the parent folders of the current folder.
    - If a folder is found with that name containing an index.js file then that index.js file is loaded.

- For syntax with path starting with ./ or ../ like: const test = require('./lib/example.js') :
    - Such syntax may contain an absolute path or a relative path. So the file present at that location is loaded in this case.

Note: If no file is found after these steps then an error is thrown.

- **Loading:**
  In this step Node loads the module and determines the type of file content.
- **Wrapping:**
  After loading, the module code is wrapped in a special function that will give access to a couple of objects. It also gives a separate scope to the variables.
- **Evaluating:**
  At this step JavaScript Engine (usually V8) executes the code present in the wrapper function and exports functions or variables mentioned with the module.exports in the file.
- **Caching:**
  After the evaluation step, Node.js modules are cached (i.e., stored for future use) when the module is loaded for the first time. And later if you need to load the same node module next time then node.js does not go through above mentioned steps for that module again as it will copy that module from the cache.

Syntax and Examples for Require in Node.js

Loading core modules

Syntax:

```
var test = require('module_name')
```

Example:

```
var http = require('http')
```

Loading local modules

For loading locally created modules, we can provide a path to the required function in the following ways.

Using absolute path

Syntax:

```
var test = require('/<folder1>/<folder2>/.../module')
```

Example:

```
var http = require('/lib/obj/util.js')
```

Using relative path

We can also provide a relative path using ./ or ../ in the required function.

Syntax:

```
var test = require('./module_name')
```

Example:

```
var hello = require('../hello.js')
```

Note:

You can omit the .js extension in the above example also i.e. if you don't provide any extension then Node searches for a file with that module_name and .js extension and loads it.

Using folder path

You can load modules just by folder path also:

```
var test = require('./folder_name')
```

Example:

```
var hello = require('./lib')
```

By default, the node finds the index.js file in that folder and loads it. Otherwise, we can also create a package.json file in that folder where we can define the node module name which we want to load by default.

Import in Node.js

As required works in the CommonJs modules system, similarly, import is used for including modules in ES6 (version 6 of the ECMA Script) module system. Which means import is used to include an ES module. At present Node.js doesn't support

ES6 import directly. So if we try to use the import keyword for importing modules directly in node js it will throw out the error. So how can we use import for including modules then you can do so in the following ways:

Using ".mjs" extension

The first way to use the ES6 import statement in Node.js is to save the JavaScript file with the ".mjs" extension, instead of using the typical ".js" extension.

As mentioned above the default module system for Node.js is CommonJs which supports require function for importing modules. So if we want to use the ECMAScript module system then the ".mjs" extension helps us achieve that.

Using package.json file

By this method, we can use the ".js" extension while using the import statement of ECMAScript. For this, we need to include a package.json file in our project. And the content of package.json should be like this:

```
// package.json
{
  "name": "node_import",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
```

```
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

The **"type"** property present in the above package.json file helps in deciding the module system that the project should use. The "type" can be **"module"** or **"commonjs"**. If the type is **"module"** then it enables the ECMAScript module system, whereas if the type is **"commonjs"** then it is of the CommonJS module system.

We can use import statements in the following ways:

```
// Importing the entire module
import * as name from 'module_name'

// Importing the default export from the module
import name from 'module_name'

// Importing a single export from the module:
import { name } from 'module_name'

// Importing multiple exports from the module:
import { name1, name2 } from 'module_name'
```

Difference between Require and Import

require	**import**
It is used in the CommonJS module system.	It is used in the ES6 (ECMAScript version 6) module system.
Loading is synchronous in require (i.e., modules are imported sequentially.)	Loading is asynchronous in import (i.e., modules are imported without waiting for previous module import to complete.)
Because of synchronous loading performance of require less efficient than import.	Because the Asynchronous loading performance of import is better than required.
If we import a module using require then the complete module is imported. So, memory usage is more.	Using import we can selectively load pieces of code in the module. So, memory usage is less compared to require.
require imports of the components exported by module. exports in the module.	import includes components exported by export in the module.
require can be called directly as it is the default way of importing.	To use import in our project we need to enable ES6 or ECMAScript module in our project.
require can be called anywhere in the program	import works only at the top of the program

Hybrid CommonJS/Node.js/ES6 modulescenarios

We've gone over the format for CommonJS/Node.js modules, the format for ES6 modules, and the algorithm for locating and importing both. The last thing to cover is those hybrid situations where our code will use both module formats at the same time.

As a practical matter, ES6 modules are very new to the Node.js platform, and therefore we have a large body of existing code written as CommonJS/Node.js modules. Many tools in the Node.js market have implementation dependencies on the CommonJS format. This means we'll be facing situations where ES6 modules will need to use CommonJS modules, and vice versa:

- CommonJS module loads other CommonJS modules with require()
- CommonJS module cannot load ES6 modules—except for two
    - methods: Dynamic import, also known as import(), can load an ES6 module as an asynchronous operation
    - The @std/esm package supplies a require() function with one that can load ES6 modules as an asynchronous operation
- ES6 modules load other ES6 modules with import, with the full semantics of the
  import statement
- ES6 modules load CommonJS modules using import

Therefore, out of the box, three of the scenarios are directly supported. The fourth is supported with a workaround module.

When an ES6 module loads a CommonJS module, its module.exports object is exposed as the default export of the module. This means your code uses this pattern:

```
import cjsModule from 'commoл–js–module';
...
cjsModule.fuлctioлName();
```

This is extremely similar to using a CommonJS module in another CommonJS module. You are simply transliterating the require() call into an import statement.

Dynamic imports with import()

ES6 modules do not cover all the requirements to fully replace Node.js/CommonJS modules. One of the missing capabilities is being addressed with the Dynamic Import feature currently on its way through the TC-39 committee.

Support for dynamic imports landed in Node.js 9.7. See the documentation at:
https://github.com/tc39/proposal–dynamic–import.

We'll use dynamic imports to solve an issue in Chapter 7, *Data Storage and Retrieval*, about dynamically choosing the module to load. In normal usage of the require() statement, can use a simple string literal to specify the module name. But it is also possible to use a string literal to compute the module name, like so:

```
// Node.js dynamically determined module loading
const moduleName = require(`../models/${process.env.MODEL_NAME}`);
```

We used this technique in earlier editions of this book to dynamically choose between several implementations of the same model API. The ES6 import statement does not support anything but a simple string literal, and therefore cannot compute the module specifier like this example.

With dynamic imports, we have an import() function where the module specifier is a regular string, letting us make a similar dynamic choice of module. Unlike the require()function, which is synchronous, import() is asynchronous, and returns a Promise. Hence, it's not a direct replacement for require() in that it's not terribly useful as a top-level function. You'll see how to use it in Chapter 7, *Data Storage and Retrieval*.

Perhaps the most important feature it brings is that CommonJS modules can use import()
to load an ES6 module.

NPM – NODE JS PACKAGE MANAGEMENT SYSTEM

Introduction to npm

NPM is the standard package manager for Node.js.

In September 2022 over 2.1 million packages were reported being listed in the npm registry, making it the biggest single language code repository on Earth, and you can be sure there is a package for (almost!) everything.

It started as a way to download and manage dependencies of Node.js packages, but it has since become a tool used also in frontend JavaScript.

Yarn and **pnpm** are alternatives to npm cli. You can check them out as well.

Packages

npm manages downloads of dependencies of your project.

Installing all dependencies

If a project has a package.json file, by running

```
npm install
```
Bash

it will install everything the project needs, in the node_modules folder, creating it if it's not existing already.

Installing a single package

You can also install a specific package by running

```
npm install <package-name>
```
Bash

Furthermore, since npm 5, this command adds <package-name> to the package.json file *dependencies*. Before version 5, you needed to add the flag --save.

Often you'll see more flags added to this command:

- --save-dev installs and adds the entry to the package.json file *devDependencies*
- --no-save installs but does not add the entry to the package.json file *dependencies*
- --save-optional installs and adds the entry to the package.json file *optionalDependencies*
- --no-optional will prevent optional dependencies from being installed

Shorthands of the flags can also be used:

- -S: --save
- -D: --save-dev
- -O: --save-optional

The difference between *devDependencies* and *dependencies* is that the former contains development tools, like a testing library, while the latter is bundled with the app in production.

As for the *optionalDependencies* the difference is that build failure of the dependency will not cause installation to fail. But it is your program's responsibility to handle the lack of the dependency. Read more about optional dependencies.

Updating packages

Updating is also made easy, by running

```
npm update
```
Bash

npm will check all packages for a newer version that satisfies your versioning constraints.

You can specify a single package to update as well:

```
npm update <package-name>
```
Bash

Versioning

In addition to plain downloads, npm also manages **versioning**, so you can specify any specific version of a package, or require a version higher or lower than what you need.

Many times you'll find that a library is only compatible with a major release of another library.

Or a bug in the latest release of a lib, still unfixed, is causing an issue.

Specifying an explicit version of a library also helps to keep everyone on the same exact version of a package, so that the whole team runs the same version until the package.json file is updated.

In all those cases, versioning helps a lot, and npm follows the semantic versioning (semver) standard.

You can install a specific version of a package, by running

```
npm install <package-name>@<version>
```
Bash

Running Tasks

The package.json file supports a format for specifying command line tasks that can be run by using

```
npm run <task-name>
```
Bash

For example:

```
{  "scripts": {    "start-dev": "node lib/server-development",    "start": "node lib/server-production"  }}
```
JSON

It's very common to use this feature to run Webpack:

```
{   "scripts": {      "watch": "webpack --watch --progress --colors --config webpack.conf.js",          "dev": "webpack --progress --colors --config webpack.conf.js",    "prod": "NODE_ENV=production webpack -p --config webpack.conf.js"  }}
```
JSON

So instead of typing those long commands, which are easy to forget or mistype, you can run

```
$ npm run watch$ npm run dev$ npm run prod
```
Bash

9. Assignment Questions

Category - 1

S.No.	Write Programs for the following	K - Level	COs
1	Use Node JS to build a Web Application	K2	CO1
2	Create Node JS Service for an Education Site	K3	CO1

Category - 2

S.No.	Write Programs for the following	K - Level	COs
1	Create an Node JS to build an welcome page	K5	CO1
2	Develop an application to demonstrate department website using REACT JS	K3	CO1

Category - 3

S.No.	Write Programs for the following	K - Level	COs
1	Develop a shopping cart using REACT JS	K5	CO1
2	Develop a Image Loading application using REACT JS	K5	CO1

10. Assignment Questions

Category - 4

S.No.	Write Programs for the following	K - Level	COs
1	Integrate REACT JS and Database	K5	CO1
2	Design a calculator using REACT JS	K5	CO1

Category - 5

S.No.	Write Programs for the following	K - Level	COs
1	Develop an REACT JS application to store and retrieve	K5	CO1
2	Sort the employee names in ascending order	K5	CO1

11. Part A
Question & Answer

Part A

1. **What is Node.js? CO1**
   Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to run JavaScript code outside the browser.

2. **Who developed Node.js? CO1**
   Node.js was developed by Ryan Dahl in 2009.

3. **What is the main advantage of using Node.js? CO1**
   The main advantage of using Node.js is its event-driven, non-blocking I/O model, which makes it lightweight and efficient for building real-time applications.

4. **How does Node.js handle concurrent requests? CO1**
   Node.js uses an event loop to handle concurrent requests asynchronously, allowing it to handle multiple connections without blocking.

5. **What is npm? CO1**
   npm (Node Package Manager) is the default package manager for Node.js, used to install, manage, and share JavaScript packages.

6. **How can you install Node.js on your system? CO2**
   Node.js can be installed by downloading the installer from the official Node.js website and following the installation instructions.

7. **What command is used to check the installed version of Node.js? CO1**
   The command `node -v` or `node --version` can be used to check the installed version of Node.js.

8. **How do you create a new Node.js project? CO2**
   You can create a new Node.js project by running `npm init` in the project directory and following the prompts to set up the project metadata.

9. **What is the purpose of the package.json file in a Node.js project? CO1**
   The package.json file in a Node.js project contains metadata about the project, including its dependencies, scripts, and other configuration settings.

10. **How do you install dependencies for a Node.js project? CO2**
    You can install dependencies for a Node.js project by running `npm install <package-name>` or `npm install` to install all dependencies listed in the package.json file.

11. **What is a Node.js module? CO1**
    A Node.js module is a JavaScript file that encapsulates a piece of functionality, making it reusable and easy to manage.

12. How do you import a module in Node.js? CO3
You can import a module in Node.js using the `require()` function, passing the path to the module file as an argument.

13. What is the difference between built-in modules and external modules in Node.js? CO1
Built-in modules are modules that are included with Node.js, such as `fs` (file system) and `http` (HTTP server). External modules are third-party modules installed via npm.

14. How do you create and export a custom module in Node.js? CO3
To create a custom module in Node.js, you create a JavaScript file containing the desired functionality and use `module.exports` to export the functions or objects you want to make available to other modules.

15. Can you use ES6 modules in Node.js? CO1
Yes, Node.js has experimental support for ES6 modules. You can use the `.mjs` file extension and use `import` and `export` statements to work with ES6 modules.

16. What is the purpose of the `npm install` command? CO1
The `npm install` command is used to install dependencies for a Node.js project based on the dependencies listed in the package.json file.

17. How do you install a specific version of a package using npm? CO3
You can install a specific version of a package using the `npm install <package-name>@<version>` syntax.

18. What is the difference between `dependencies` and `devDependencies` in package.json? CO1
Dependencies listed under `dependencies` are required for the application to run, while dependencies listed under `devDependencies` are only required for development purposes (e.g., testing frameworks, build tools).

19. How do you uninstall a package using npm? CO3
You can uninstall a package using the `npm uninstall <package-name>` command.

20. What is the purpose of the `npm start` command? CO1
The `npm start` command is used to start the application based on the `"start"` script defined in the package.json file.

21. Can you mix CommonJS and ES6 modules in a Node.js project? CO4
    Yes, you can mix CommonJS and ES6 modules in a Node.js project, but you need to be cautious about compatibility issues and ensure proper module resolution.

22. How do you import a CommonJS module into an ES6 module? CO3
    You can import a CommonJS module into an ES6 module using `require()` syntax, but you may need to use a transpiler like Babel to convert CommonJS modules to ES6 modules.

23. What are the benefits of using ES6 modules over CommonJS modules? CO1
    ES6 modules offer better syntax, support for static analysis and tree-shaking, and compatibility with browser-based JavaScript environments.

24. How do you export an ES6 module as a CommonJS module? CO3
    You can export an ES6 module as a CommonJS module by assigning the module's exports to `module.exports`.

25. What are some potential challenges when working with hybrid module scenarios in Node.js? CO1

    Potential challenges include compatibility issues between CommonJS and ES6 modules, differences in module resolution, and the need for additional tooling such as transpilers.

12.PART – B Questions

.

1. Explain the concept of event-driven programming in Node.js. Provide an example scenario where event-driven architecture is beneficial. (CO 2)

2. Compare and contrast the performance characteristics of synchronous and asynchronous I/O operations in Node.js. Provide examples to illustrate each type of operation. (CO 3)

3. Discuss the role of the Node Package Manager (npm) in the Node.js ecosystem. Explain how npm facilitates package management and dependency resolution in Node.js projects. (CO 2)

4. Describe the purpose and functionality of the `package.json` file in a Node.js project. Provide examples of key fields typically found in the `package.json` file and explain their significance. (CO: 2)

5. Explain the difference between the `dependencies` and `devDependencies` sections in a `package.json` file. Provide examples of scenarios where each type of dependency is commonly used in Node.js projects. (CO: 2)

13. Supportive online courses

Online courses

1. https://www.udemy.com/topic/spring-boot/
2. https://www.guvi.in/mlp/join-full-stack-program
3. https://www.codingninjas.com/careercamp/professionals/
4. https://www.coursera.org/learn/spring-repositories
5. https://www.coursera.org/learn/google-cloud-java-spring

External Links for Additional Resources

1. https://spring.io/guides/gs/spring-boot/
2. https://www.baeldung.com/spring-boot-start
3. https://www.interviewbit.com/spring-boot-interview-questions/

14.Real Time Applications

E-commerce Platform:
Social Media Dashboard
Collaborative Document Editing

14. Content Beyond Syllabus

Sending and receiving events with Event Emitters

Event Emitters are one of the core idioms of Node.js. If Node.js's core idea is an event-driven architecture, emitting events from an object is one of the primary mechanisms of that architecture. An Event Emitter is an object that gives notifications events at different points in its life cycle. For example, an HTTP Server object emits events concerning each stage of the startup/shutdown of the Server object, and as HTTP requests are made from Many core Node.js modules are EventEmitters, and EventEmitters are an excellent skeleton to implement asynchronous programming. EventEmitters have nothing to do with web application development, but they are so much part of the Node.js woodwork that you may skip over their existence.

The EventEmitter Class

The EventEmitter object object is defined in the events module of Node.js. Directly using the EventEmitter class means performing require ('events'). In most cases, you'll be using an existing object that uses EventEmitter internally and you won't require this module. But there are cases where needs dictate implementing an EventEmitter Subclass.

15. Assessment Schedule

Tentative schedule for the Assessment During 2023-2024 Even Semester

S. No.	Name of the Assessment	Start Date	End Date	Portion
1	Unit Test 1			Unit 1
2	IAT 1	12.02.2024	17.02.2024	Unit 1 & 2
3	Unit Test 2			Unit 3
4	IAT 2	01.04.2024	06.04.2024	Unit 3 & 4
5	Revision 1			Unit 5, 1 & 2
6	Revision 2			Unit 3 & 4
7	Model	20.04.2024	30.04.2024	All 5 Units

16. Text Books & References

TEXT BOOKS:

1. Somnath Musib, Spring Boot in Practice, Manning publication, June 2022 (https://www.manning.com/books/spring-boot-in-practice)
2. Alex Banks, Eve Porcello , "Learning React", May 2017, O'Reilly Media, Inc. ISBN: 9781491954621. (https://www.oreilly.com/library/view/learning-react/9781491954614/)
3. David Herron ,"Node.js Web Development - Fourth Edition",2018, Packt Publishing,ISBN: 9781788626859
4. Sukesh Marla, "A Journey to Angular Development Paperback ", BPB Publications.(https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r)
5. Yakov Fain Anton Moiseev, "Angular Development with TypeScript", 2nd Edition.(https://www.manning.com/books/angular-development-with-typescript-Second Edition).

Reference Books:

REFERENCES:

1. Sue Spielman, The Struts Framework 1: A Practical guide for Java Programmers‖, 1st Edition. Elsevier 2002

WEB REFERENCES:

1. https://www.manning.com/books/spring-boot-in-practice
2. https://www.oreilly.com/library/view/learning- react/9781491954614
3. https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r
4. https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r
5. https://www.manning.com/books/angular-development-with-typescript-Second Edition

17. Mini Project Suggestions

To Do App

A app that keeps track of your to-do actions, you can add any number of tasks to it, search those tasks, click on the checkbox to complete the task, and filter using buttons like active tasks, completed tasks and all the tasks. This is another great project for you to add to your portfolio as a beginner React developer.

E Commerce

An e-commerce website built using React and Typescript, where we can filter clothes products using their customer preferred sizes as M, L or XL etc. We have a button called “Add to cart” below each product shown on the web page, once user selects any product, it will go to cart. At the end it can be used to checkout. These terms must be familiar to everyone now-a-days since it tries to mock popular e- commerce websites like Amazon, Flipkart and Myntra etc.

Category -2

1. **To-Do List App**: Create a simple to-do list app with features to add, mark as done, and delete tasks.
2. **Counter App**: Build a counter application that increments or decrements a value when buttons are clicked.

Category -3

1. **Calculator**: Develop a basic calculator that performs arithmetic operations on user inputs.
2. **Random Quote Generator**: Create an app that displays random quotes fetched from an API.

18. Mini Project Suggestions

Category – 4

3. **Weather App**: Build an app that fetches weather data based on user input location.
4. **Image Gallery**: Design a gallery where users can view and search for images.

Category – 5

8. **Timer/Stopwatch**: Build a timer or stopwatch application with start, pause, and reset functionalities.
9. **Simple Blog**: Create a basic blog platform where users can read and write posts.

Thank you

R.M.K GROUP OF INSTITUTIONS