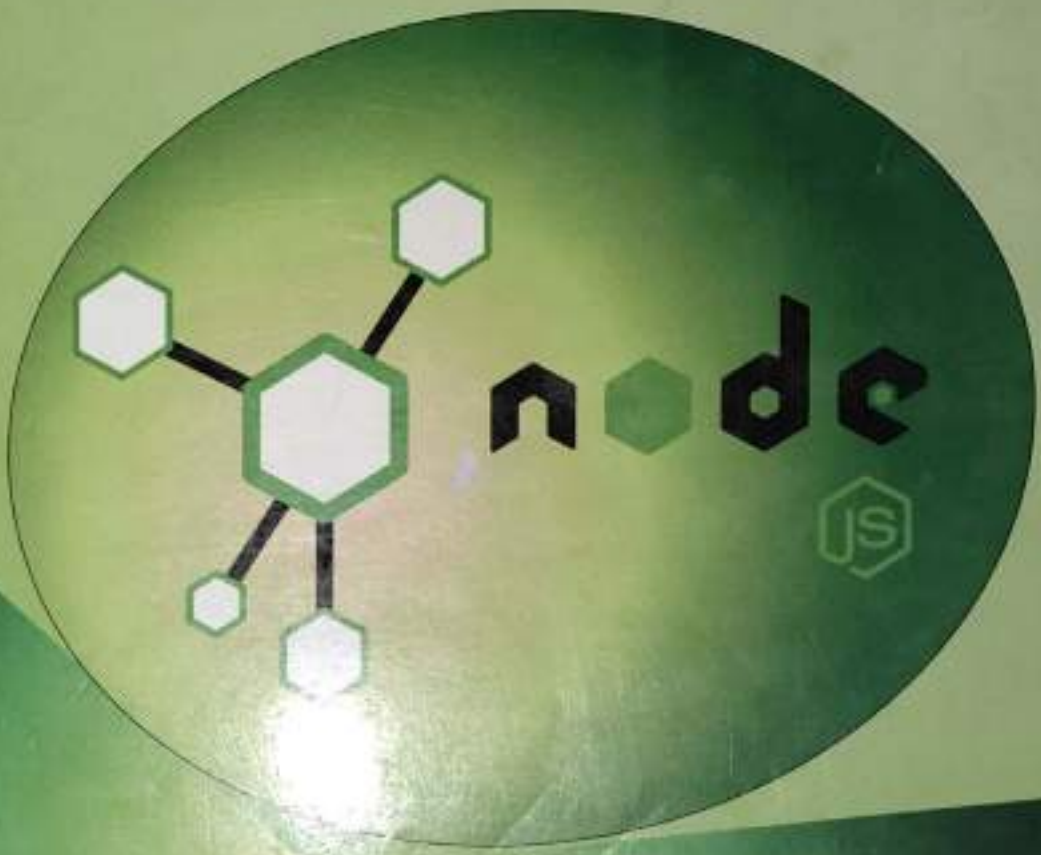
NEW SYLLABUS
CBCS PATTERN

S.Y. B.B.A. (C.A.)
SEMESTER - IV

NODE-JS

SHIVENDU BHUSHAN
PRADIP WAGHMARE



NIRALI
PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

Azim. Aaga

SPPU New Syllabus

A Book Of

NODE - JS

For BBA (Computer Applications): Semester - IV

[Course Code CA - 404 [Option]]

CBCS Pattern

As Per New Syllabus, Effective from June 2020

Mr. Shivendu Bhushan

M.C.A,
Vice Principle and HOD of BBA (CA) Deptt.,
Indira College of Commerce and Science,
Pune.

Mr. Pradip Waghmare

M.C.A. Pune.

Price ₹ 150.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N4957

Node - JS

ISBN 978-93-90596-35-

First Edition : January 2021

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39, Fax - (020) 25511379
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 41104
Mobile No. 9404233041/9850046155

DISTRIBUTION CENTRES

PUNE

Nirali Prakashan (For orders within Pune) : 119, Budhwar Peth, Jogeshwari Mandir Lane, Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

Nirali Prakashan (For orders outside Pune) : S. No. 28/27, Dhayari, Near Asian College Pune 411041
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

MUMBAI

Nirali Prakashan : 385, S.V.P. Road, Rasdhara Co-op. Hsg. Society Ltd.,
Girgaum, Mumbai 400004, Maharashtra; Mobile : 9320129587
Tel : (022) 2385 6339 / 2386 9976, Fax : (022) 2386 9976
Email : niralimumbai@pragationline.com

DISTRIBUTION BRANCHES

JALGAON

Nirali Prakashan : 34, V. V. Golani Market, Navi Peth, Jalgaon 425001, Maharashtra,
Tel : (0257) 222 0395, Mob : 94234 91860; Email : niralijalgaon@pragationline.com

KOLHAPUR

Nirali Prakashan : New Mahadvar Road, Kedar Plaza, 1st Floor Opp. IDBI Bank, Kolhapur 416 012
Maharashtra. Mob : 9850046155; Email : niralikolhapur@pragationline.com

NAGPUR

Nirali Prakashan : Above Maratha Mandir, Shop No. 3, First Floor,
Rani Jhanshi Square, Sitabuldi, Nagpur 440012, Maharashtra
Tel : (0712) 254 7129; Email : niralinagpur@pragationline.com

DELHI

Nirali Prakashan : Room No. 2, Ground Floor, 4575/15 Onkar Tower, Aggarwal Road, Daryaganj
New Delhi 110002 Mob : +91 9555778814 / +91 9818561840
Email : niralidelhi@pragationline.com

BENGALURU

Nirali Prakashan : Maitri Ground Floor, Jaya Apartments, No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003, Karnataka; Mob : 9449043034
Email: niralibangalore@pragationline.com

Other Branches : Hyderabad, Chennai

Preface ...

We take this opportunity to present this book entitled as "**Node - JS**" to the students of B.B.A (CA) - Fourth Semester. The object of this book is to present the subject matter in a most concise and simple manner. The book is written strictly according to the New Syllabus (CBCS Pattern).

The book has its own unique features. It brings out the subject in a very simple and lucid manner for easy and comprehensive understanding of the basic concepts, its intricacies, procedures and practices. This book will help the readers to have a broader view on Node - JS. The language used in this book is easy and will help students to improve their vocabulary of Technical terms and understand the matter in a better and happier way.

We sincerely thank Shri. Dineshbhai Furia and Shri. Jignesh Furia of Nirali Prakashan, for the confidence reposed in us and giving us this opportunity to reach out to the students as well as teachers.

We have given our best inputs for this book. Any suggestions towards the improvement of this book and sincere comments are most welcome on niralipune@pragationline.com.

Authors

Syllabus ...

1. Network Models Introduction to Node JS **(8 Hrs.)**

1.1 Introduction
1.2 What is Node JS?
1.3 Advantages of Node JS
1.4 Traditional Web Server Model
1.5 Node.js Process Model
1.6 Install Node.js on Windows
1.7 Working in REPL

2. Node JS Modules **(10 Hrs.)**

2.1 Functions
2.2 Buffer
2.3 Module
2.4 Module Types
2.5 Core Modules
2.6 Local Modules
2.7 Module.Exports

3. Node Package Manager **(6 Hrs.)**

3.1 What is NPM ?
3.2 Installing Packages Locally
3.3 Adding dependency in package.json
3.4 Installing packages globally
3.5 Updating packages

4. Web Server **(6 Hrs.)**

4.1 Creating web server
4.2 Handling http requests
4.3 Sending requests

5. File System **(8 Hrs.)**

5.1 Fs.readFile
5.2 Writing a File
5.3 Writing a file asynchronously
5.4 Opening a file
5.5 Deleting a file
5.6 Other IO Operations

6. Events **(4 Hrs.)**

6.1 EventEmitter class
6.2 Returning event emitter
6.3 Inhering events

7. Database Connectivity **(6 Hrs.)**

7.1 Connection string
7.2 Configuring
7.3 Working with select command
7.4 Updating records
7.5 Deleting records

Contents ...

❖❖❖

1... Introduction to Node JS

Objectives...

- To learn about Node JS.
- To study Different advantages of Node JS.
- To study about Traditional Web Server Model.
- To learn Node JS Process Model.
- To understand how to install Node JS on Windows.
- To learn Working in REPL.

1.1 INTRODUCTION

- The modern web application has really come a long way over the years with the introduction of many popular frameworks such as bootstrap, Angular JS, etc. All of these frameworks are based on the popular JavaScript framework. But when it came to developing server based applications there was just kind of a void, and this is where Node JS came into the picture.
- Earlier the internet programming was divided in two phases:
  1) **Client Side Scripting:** Client side scripting is performed to generate a code that can run on the client end (browser) without needing the server side processing. Basically, these types of scripts are placed inside an HTML document.
  2) **Server Side Scripting:** Server side scripting is a technique of programming for producing the code which can run software on the server side. In simple words, any scripting or programming that can run on the web server is known as Server Side Scripting.

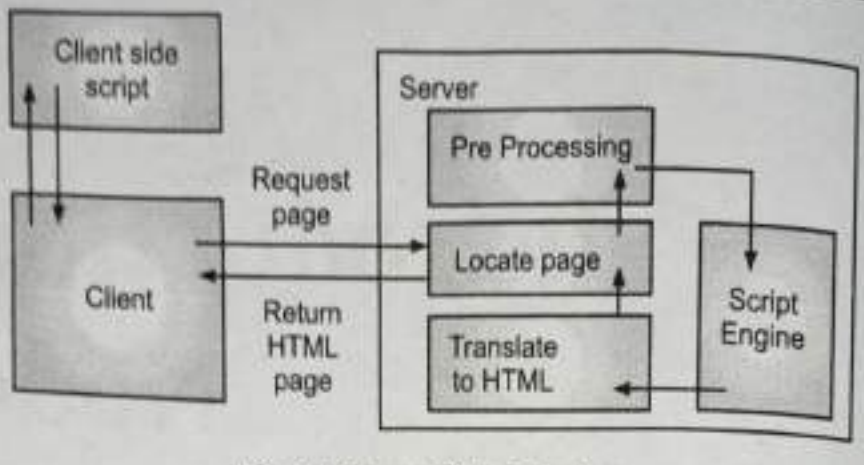


Fig. 1.1: Server Side Scripting

- Difference Between Client Side Scripting and Server Side Scripting:

Table 1.1: Difference between Client Side Scripting and Server Side Scripting

Sr. No.	Client Side Scripting	Server Side Scripting
1.	Web browsers execute client side scripting.	Web servers are used to execute server side scripting.
2.	It is also used for validations and functionality for user events.	They are basically used to create dynamic pages.
3.	It cannot be basically used to connect to databases on web server.	It cannot be basically used to connect to databases on web server.
4.	These scripts cannot access file system that resides at web server.	It can also access the file system residing at web server.
5.	It can also used to create "cookies" that store data on user's computer.	Server side environment that runs on a scripting language is a web server.

Web Based Scripting Language:

- Since inception of web development, the client side scripting languages like VB Script Java Script etc. and server side scripting languages like ASP, JSP, PHP etc. was different. At that time, developer was learning both for client end and server end. Till few years back Java Script was limited to client side scripting only.
- But in year 2009, an idea came in the mind of Ryan Dahl, a Google Engineer that, why not one language should work at both end client end as well as at server end. So he worked to run JavaScript outside client browser also. Means same JavaScript could be used at client end as well as at server end. So he used tool Chrome V8 engine and embedded in a C++ program and called it Node.exe which later on became Node JS.

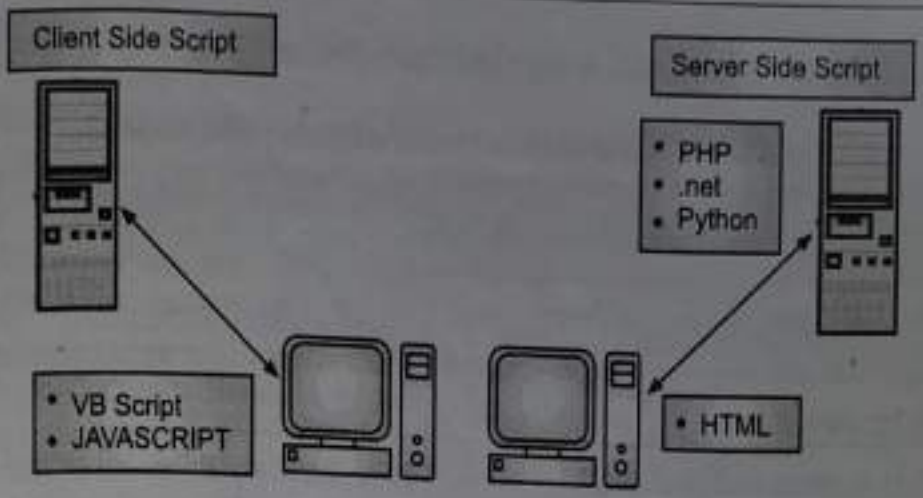


Fig. 1.2: Web Based Scripting Language

- Node JS is also based on the JavaScript framework, but it is used for developing server based applications. Node JS is not a framework and it's not a programming language also.
- We often use Node JS for building back end services like APIs, Web App or Mobile App. It is used in production by large companies such as PayPal, Uber, Netflix, Walmart and so on.

1.2 WHAT IS NODE JS?

- A Node JS is a single process application; it does not create a new thread for every request.
- Node JS runs the V8 JavaScript engine, which is the core of Google Chrome browser.
- Node JS is an open source and cross platform JavaScript runtime environment. It is very easy and popular tool for developing any kind of websites.
- Node JS provides a collection of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node JS are written using non-blocking paradigms, making blocking behavior the exception.
- When Node JS performs an I/O operation, like reading from the network, accessing a database or the file system, instead of blocking the thread and wasting CPU cycles waiting, Node JS will resume the operations when the response comes back.
- This allows Node JS to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency.
- Node JS has a unique advantage because millions of frontend developers that write JavaScript for the browser are now able to write the server side code in addition to the client side code without the need to learn a completely different language.

1.2.1 Difference between Angular and Node JS

Table 1.2: Difference between Angular and Node JS

Sr. No.	Angular	Node JS
1.	It is an open source web application development framework.	It is a cross platform runtime environment for applications.
2.	It is written in TypeScript.	It is written in **C, C++** and JavaScript languages.
3.	Used for building single page client side web applications.	Used for building fast and scalable server side networking applications.
4.	Angular itself is a web application framework.	Node JS has many different frameworks like Sails.js, Partial.js, and Express.js, etc.
5.	Ideal for creating highly active and interactive web apps.	Ideal for developing small size projects.
6.	Helpful in splitting an app into MVC components.	Helpful in generating database queries.
7.	Suitable for developing real-time applications.	Suitable in situations where something faster and more scalable is required.

1.2.2 Difference between JavaScript and Node JS

Table 1.3: Difference between JavaScript and Node JS

Features	JavaScript	Node JS
1. Type	Programming Language.	Interpreter and environment for JavaScript.
2. Utility	Used for any client-side activity for a web application.	Used for accessing or performing any non-blocking operation of any operating system.
3. Running Engine	Spider monkey (Firefox), JavaScript Core (Safari), V8 (Google Chrome), etc.	V8 (Google Chrome)

1.2.3 Difference Between 'Front End' and 'Back End' Development

Table 1.4: Difference between Front End Development and Back End Development

Sr. No.	Front End Development	Back End Development
1.	Uses mark up and web languages like HTML, CSS, and JavaScript.	Uses programming and scripting languages like Python, Ruby, Perl, etc.
2.	Based on asynchronous requests and AJAX.	Based on Server Architecture.
3.	Better Accessibility.	Enhanced Security.
4.	Used for SEO.	Used for Backup.

1.2.4 Features of Node JS

1. Node JS is Asynchronous or Non-blocking nature.
2. Node JS is easy to launch and can be used for prototyping and agile development.
3. Node JS provides fast and highly scalable services.
4. It uses JavaScript everywhere so it's easy for a JavaScript programmer to build back end services using Node.js.
5. Source code cleaner, consistent and steady.
6. Large environment for open source library.

1.2.5 Application of Node JS

- Node JS is best for usage in streaming or event based real-time applications like:

1. **Online vehicle booking and tracking system:** UBER was one of the first 3 companies that put Node JS into full production. They required an extremely fast and scalable cross platform technological solution that could handle an enormous amount of notifications and requests.
2. **Chat applications.**
3. **Game servers:** Fast and high performance servers that need to processes thousands of requests at a time, then this is an ideal framework.
4. **Social media:** It is virtually impossible without mobile use. According to Statista portal, LinkedIn had an average of 63M unique members using their mobile applications, which accounted for 59% of all unique members during the same 2016.
5. **Good for Collaborative Environment:** This is good for environments which manage document. In document management environment you will have

multiple people who post their documents and do constant changes by checking out and checking in documents.

6. **Advertisement Servers:** Again here you could have thousands of requests to pull advertisements from the central server and Node JS can be an ideal framework to handle this.
7. **Streaming Servers:** Another ideal scenario to use Node is for multimedia streaming servers wherein clients have request's to pull different multimedia contents from this server.

- Node JS is good when you need high levels of concurrency but less amount of dedicated CPU time.
- Best of all, since Node JS is built on JavaScript, it's best suited when you build client side applications which are based on the same JavaScript framework.

1.2.6 Who Uses Node JS

- Node JS is used by a variety of large companies. Below is a list of a few of them.
  - **PayPal:** A lot of sites within PayPal have also started the transition onto Node.js.
  - **Yahoo, LinkedIn** is using Node JS to power their Mobile Servers, which powers the iPhone, Android, and Mobile Web products.
  - **Mozilla** has implemented Node JS to support browser APIs which has half a billion installs.
  - **EBay** hosts their HTTP API service in Node JS.

1.3 ADVANTAGES OF NODE JS

1. **Fast processing and event based model:** V8 engine used in Node JS implementation was originally developed for the Chrome browser which is written in C++. Chrome's V8 is used to compile functions written in JavaScript into machine code, and it does the job at an impressive speed. Non-blocking input output and asynchronous request handling made Node JS capable of processing request without any delay. The implementation of event based model is easy common language for both server side as well as client side. Using common language synchronization happens fast which helps time applications and event based applications.
2. **Scalability:** Developers prefer to use Node JS because it is easily scales the application in both horizontal and vertical direction.
3. **Real time web apps:** Node JS is much more convenient for chat apps or gaming apps because of faster synchronization. Also, event loop avoids HTTP overload for Node JS development.

4. **Advantage of Caching:** It provides the caching of single module. Whenever there is any request for the first module, it gets cached in the application memory so you don't need to re-execute the code.
5. **Easy to learn and code:** Node JS is easy to learn and code because it uses JavaScript. If you are a front end developer and have a good grasp on JavaScript you can easily learn and build the application on Node JS.
6. **Hosting:** PaaS (Platform as a Service) and Heroku are the hosting platform for Node JS application deployment which is easy to use without facing any issue.
7. **Data Stream:** In Node JS HTTP request and response are considered as two separate events. They are data stream so when you process a file at the time of loading it will reduce the overall time and will make it faster when the data is presented in the form of transmissions. It also allows you to stream audio and video files at lightning speed.
8. **Corporate Support:** Most of the well known companies like Walmart, PayPal, Microsoft, Yahoo are using Node JS for building the applications. Node JS uses JavaScript so most of the companies are combining front end and back end teams together into a single unit.

1.4 TRADITIONAL WEB SERVER MODEL

- In the traditional web server model, each request is handled by a dedicated thread from the thread pool.
- If no thread is available in the thread pool at any point of time then the request waits till the next available thread.
- Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.

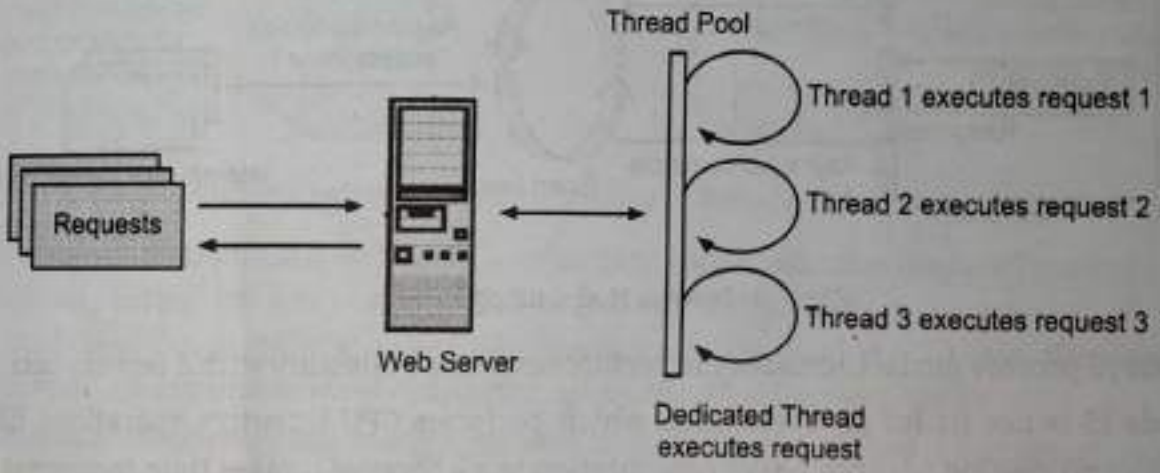


Fig. 1.3: Traditional Web Server Model

1.5 NODE JS PROCESS MODEL

- Node JS processes user requests differently as compared to a traditional web server model.
- Node JS runs in a single process and the application code runs in a single thread and thereby needs less resource than other platforms.
- All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request. So, this single thread doesn't need to wait for the request to complete, it is free to handle the next request.
- When asynchronous I/O work completes then it processes the request further and sends the response.
- An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes. Internally, Node JS uses **libev** for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.
- The following figure illustrates asynchronous web server model using Node.js.

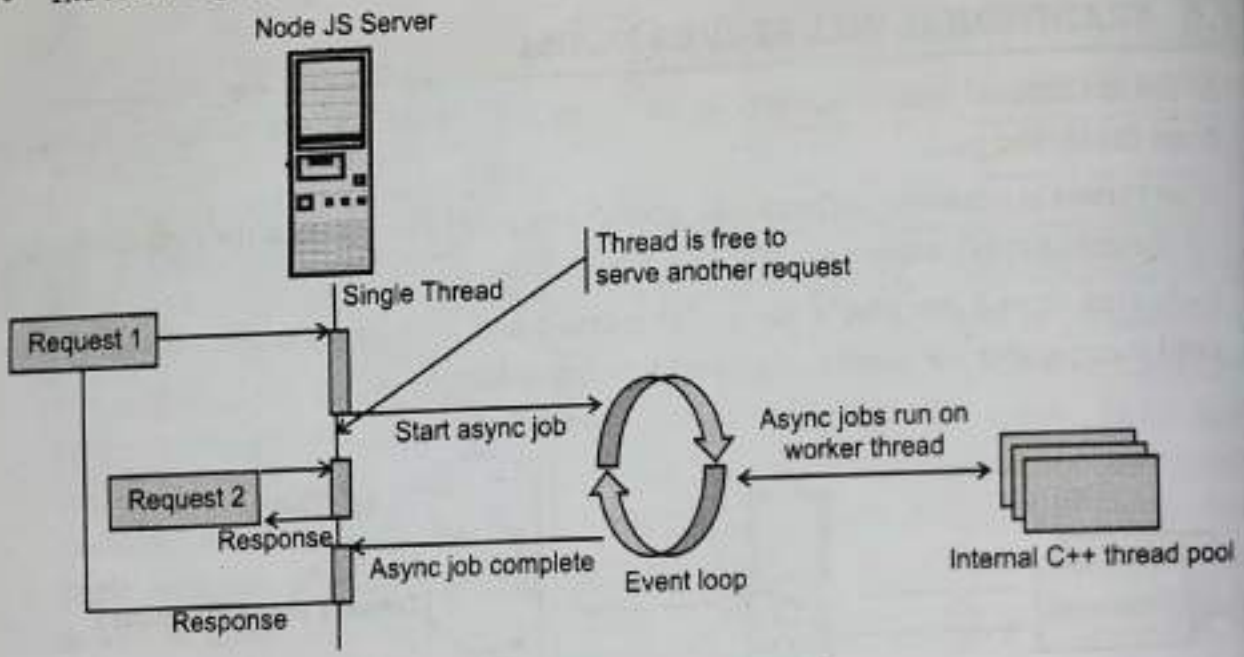


Fig. 1.4: Process Model using Node.js

- Node JS process model increases the performance and scalability with a few caveats.
- Node JS is not fit for an application which performs CPU intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.

- Node JS comes with a built-in library that allows applications to act as a web server. Node JS event driven architecture and a non-blocking I/O API optimizes an application's throughput, Node JS excels when it comes to real-time communication.

Working of Node JS:

- Node JS is a virtual machine that uses JavaScript as its scripting language and runs on a v8 environment. It works on a single threaded event loop and a non-blocking I/O which provides high rate as it can handle a higher number of concurrent requests. Also, by making use of the 'HTTP' module, Node JS can run on any stand-alone web server.

Node JS is Single Threaded:

- Node JS uses a single threaded model in order to support asynchronous processing. With asynchronous processing, an application can perform better and is more scalable under web traffic. Thus, Node JS makes use of a single threaded model approach rather than typical thread based implementation.

1.5.1 Comparison between Request Handling in Node JS and a Traditional Server

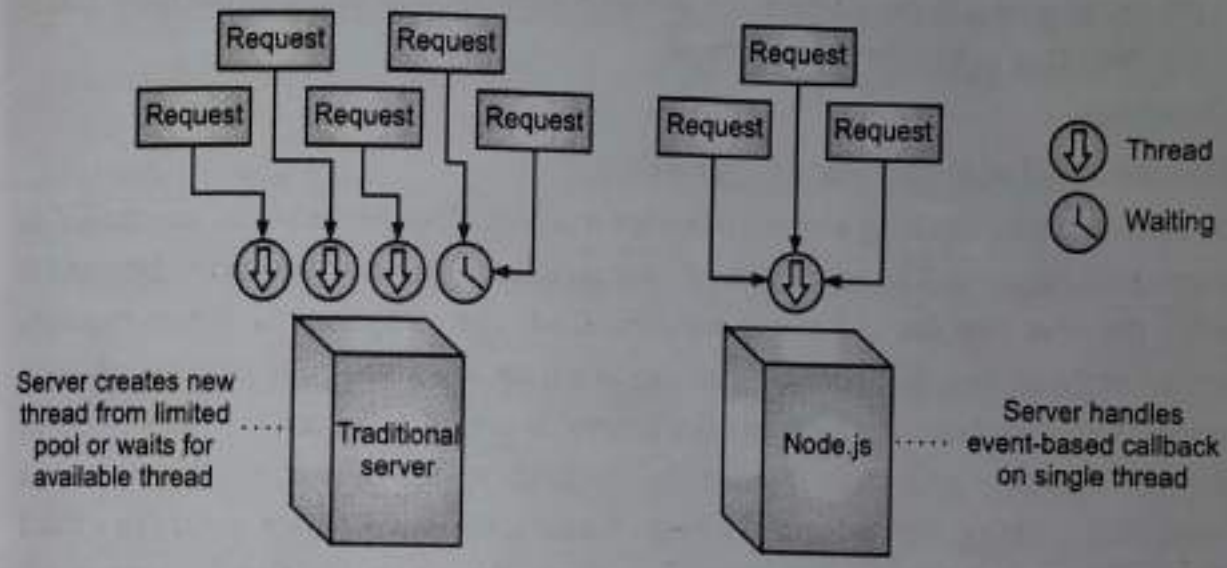


Fig. 1.5: Comparison between request handling in Node JS and a Traditional server

- Compared to traditional web-serving techniques each connection (request) spawns a new thread, taking up system RAM and eventually maxing-out at the amount of RAM available. Node.js operates on a single-thread, using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections held in the event loop.
- A quick calculation: assuming that each thread potentially has an accompanying 2 MB of memory with it, running on a system with 8 GB of RAM puts us at a theoretical

maximum of 4000 concurrent connections, plus the cost of context switching between threads. That is the scenario you typically deal with in traditional web server techniques. By avoiding all that, Node JS achieves scalability levels of over 1M concurrent connections.

1.6 BLOCKING AND NON-BLOCKING APPROCH

Blocking:

- It is also called as Synchronous Approach.
- An example of blocking is how some web servers like ones in Java or PHP handle requests. If your code does something blocking, like reading something from the database, your code "stalls" at that line and waits for the operation to finish. In that period, your machine is holding onto memory and processing time for a thread that is not doing anything. In order to cater other requests while that thread has stalled depends on your setup.
- Your server can spawn more threads to cater the request or, if you have a load balancing setup, forwards requests to the next available instance. This instills more setup, more memory consumed, more processing.
- Another Example, Imagine the situation of a restaurant. The waiter will take order from table1 give it to the kitchen and will wait until the chef prepares the food. This is called **blocking or synchronous nature.**

Non-blocking:

- It is also called as Asynchronous Approach.
- In contrast, non-blocking servers like ones made in Node JS, only use one thread to service all requests. This might sound counter-intuitive, but the creators designed it with the idea that the I/O is the bottleneck i.e. not computations. When requests arrive at the server, they are serviced one at a time. When the code serviced needs to query the DB for example, it sends off a request to the DB. However, instead of waiting for the response and stall, it sends the callback to a second queue and the code continues running. Now when the DB returns data, the callback gets queued in a third queue where they are pending execution. When the engine is doing nothing (stack empty), it picks up a callback from the third queue and executes it.
- Another Example, in contrast the waiter takes order from table1 and gives the order in the kitchen, now the waiter can serve another table let's say table2 and get their order. This is **asynchronous system**, as the chef is preparing the food, meanwhile a single waiter is serving different tables (taking orders and serving food). The waiter doesn't have to wait for the chef to cook the meal before he has to go to another table. This is what we call a **non-blocking** or **asynchronous** architecture.

1.7 FRAMEWORKS AND TOOLS OF NODE JS

- Node JS is a low level platform. To make things easy, exciting and simple for developers, thousands of libraries were built upon Node JS by the community.
- A Node JS framework is just some abstract design, built out of Node JS that represents the control flow of the given framework's design. So it is almost like the skeleton of a program.
- Many of those established over time as popular options. Here is a non comprehensive list of the ones worth learning:
  - **Meteor:** An incredibly powerful full stack framework, powering you with an isomorphic approach to building apps with JavaScript, sharing code on the client and the server.
  - **AdonisJs:** A full stack framework highly focused on developer ergonomics, stability, and confidence. Adonis is one of the fastest Node JS web frameworks.
  - **hapi:** A rich framework for building applications and services that enables developers to focus on writing reusable application logic instead of spending time building infrastructure.
  - **Fastify:** A web framework highly focused on providing the best developer experience with the least overhead and powerful plugin architecture. Fastify is one of the fastest Node JS web frameworks.
  - **Gatsby:** A React based, GraphQL powered, static site generator with a very rich ecosystem of plugins and starters.
  - **Micro:** It provides a very lightweight server to create asynchronous HTTP micro services.
  - **Loopback.io:** Makes it easy to build modern applications that require complex integrations.
  - **Nest.JS:** A Type Script based progressive Node JS framework for building enterprise grade efficient, reliable and scalable server side applications.
  - **Nx:** A toolkit for full stack monorepo development using NestJS, Express, React, Angular, and more. Nx helps scale your development from one team building one application to many teams collaborating on multiple applications.
  - **Sapper:** Sapper is a framework for building web applications of all sizes, with a beautiful development experience and flexible file system based routing. Offers SSR and more.
  - **Socket.io:** A real-time communication engine to build network applications.

- **Express:** It provides one of the simplest yet powerful ways to create a web server. Its minimalist approach, unopinionated, focused on the core features of a server, is the key to its success.
- **Next.js:** A framework to render server side rendered React applications.
- **Strapi:** Strapi is a flexible, open source Headless CMS that gives developers the freedom to choose their favorite tools and frameworks while also allowing editors to easily manage and distribute their content. By making the admin panel and API extensible through a plugin system, Strapi enables the world's largest companies to accelerate content delivery while building beautiful digital experiences.

1.8 INSTALL NODE JS ON WINDOWS

- To start building your Node JS applications, the first step is the installation of the Node JS framework. The Node JS framework is available for a variety of operating systems right from Windows to Ubuntu and OS X. Once the Node JS framework is installed you can start building your first Node JS applications.
- Node JS also has the ability to embedded external functionality or extended functionality by making use of custom modules. These modules have to be installed separately. An example of a module is the MongoDB module which allows you to work with MongoDB databases from your Node JS application.

How to Install Node JS?

- To perform the installation of Node JS, perform the below steps:

Step 1: Go to the site `https://nodejs.org/en/download/` and download the necessary binary files. In our example, we are going to the download the 32-bit setup files for Node JS.

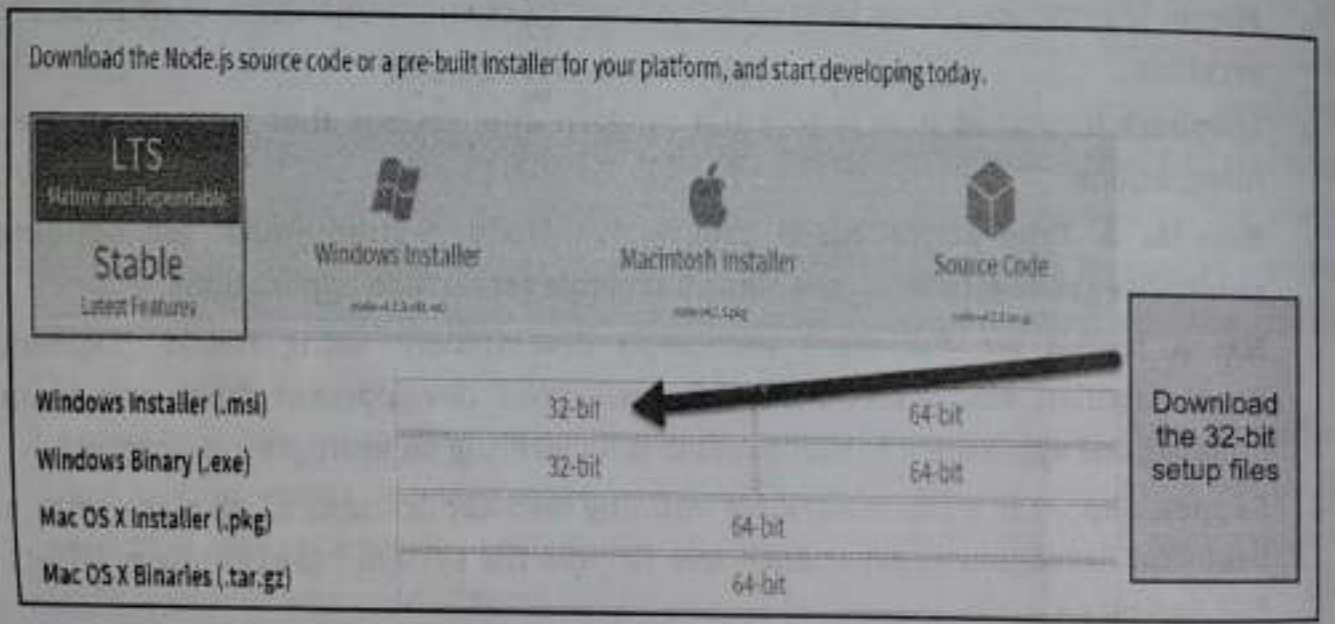


Fig. 1.6

Step 2: Double click on the downloaded .msi file to start the installation. Click the Run button in the first screen to begin the installation.

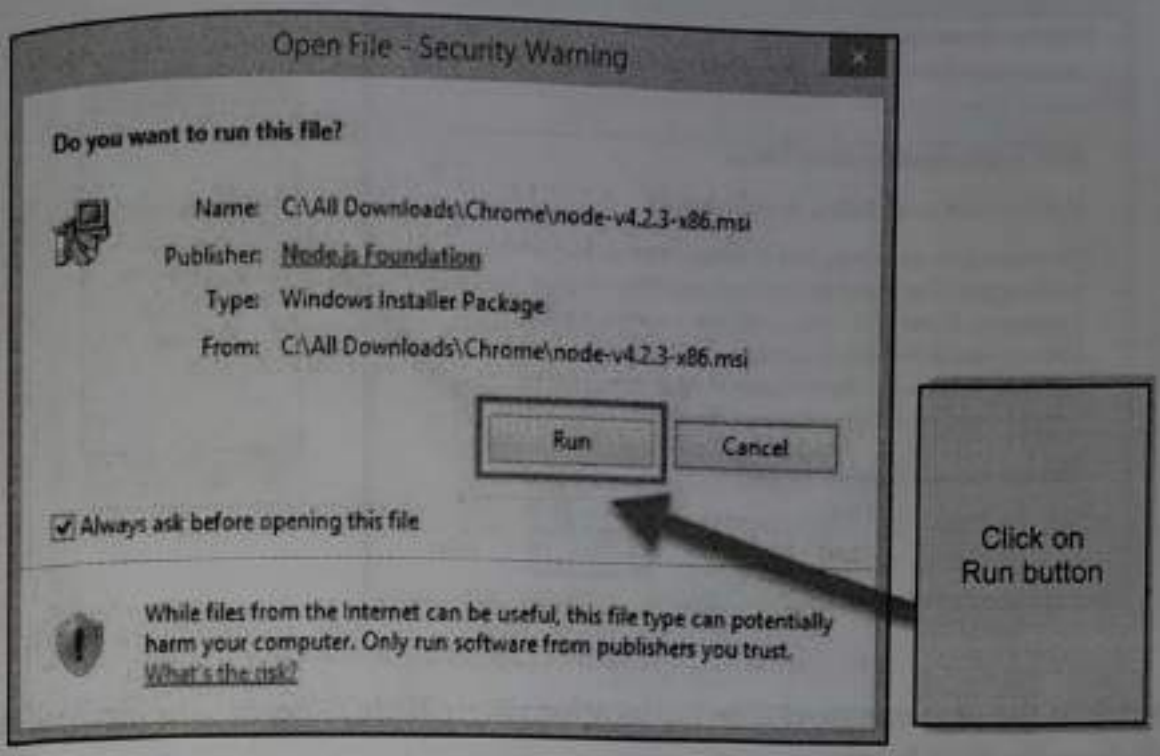


Fig. 1.7

Step 3: In the next screen, click the "Next" button to continue with the installation.

Fig. 1.8

Step 4: In the next screen, accept the license agreement and click on the Next button.

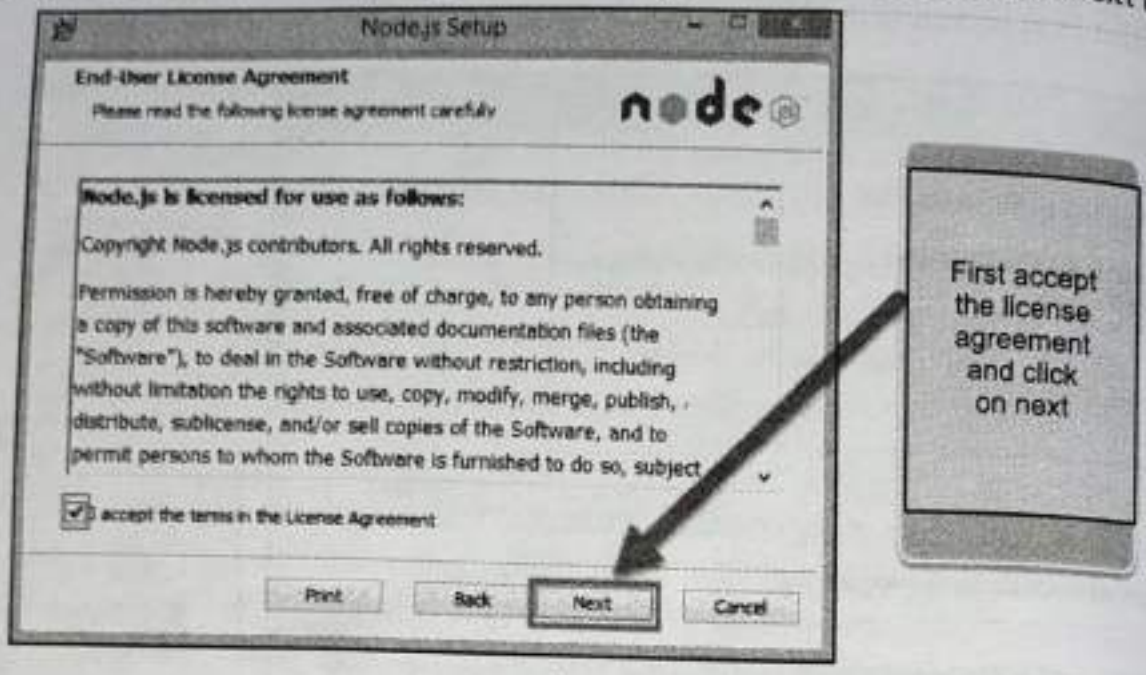


Fig. 1.9

Step 5: In the next screen, choose the location where Node JS needs to be installed and then click on the Next button.

1. First enter the file location for the installation of Node JS. This is where the files for Node JS will be stored after the installation.
2. Click on the Next button to proceed ahead with the installation.

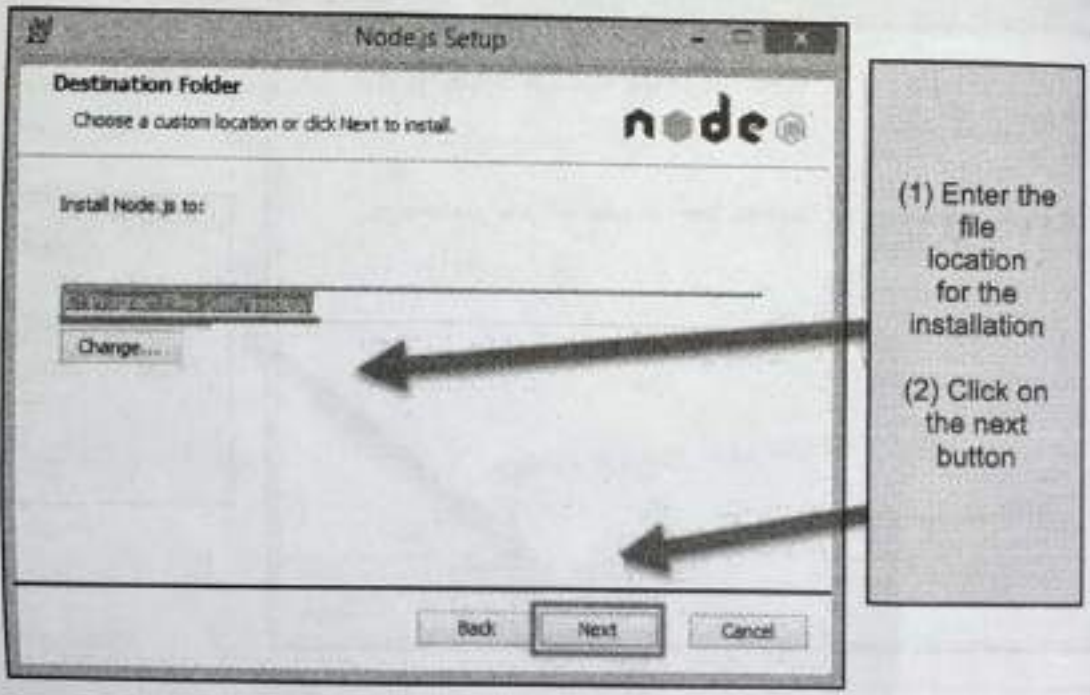


Fig. 1.10

Step 6: Accept the default components and click on the next button.

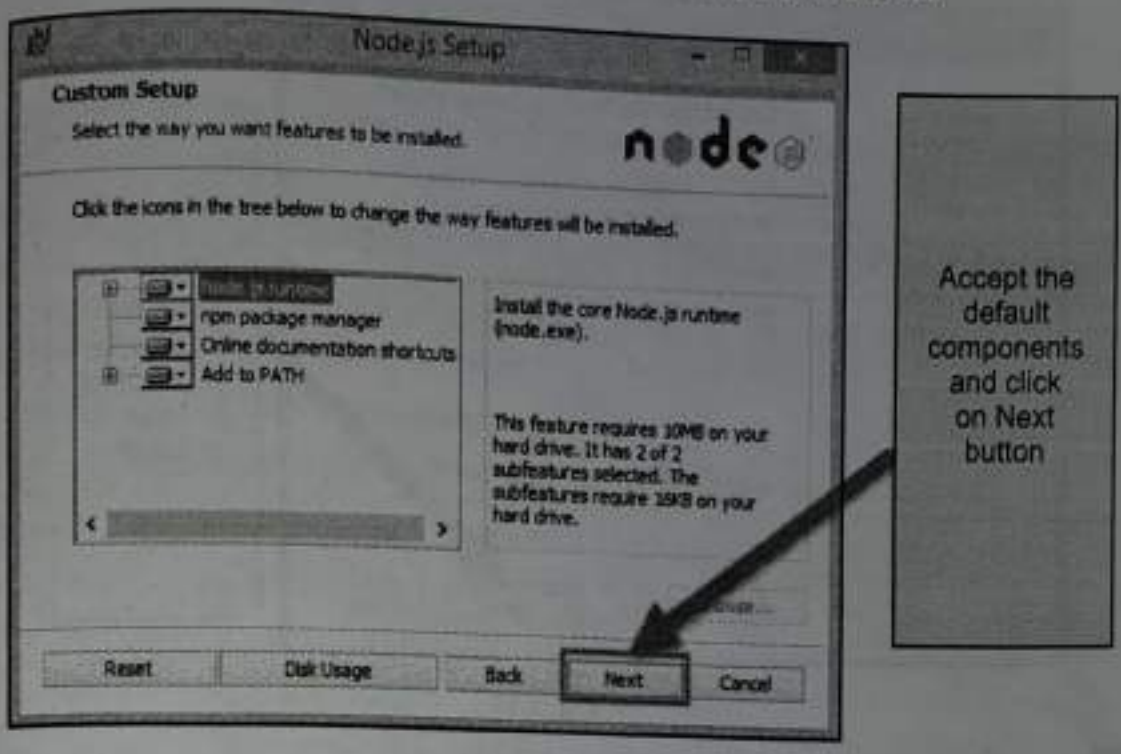


Fig. 1.11

Step 7: In the next screen, click the Install button to start the installation.

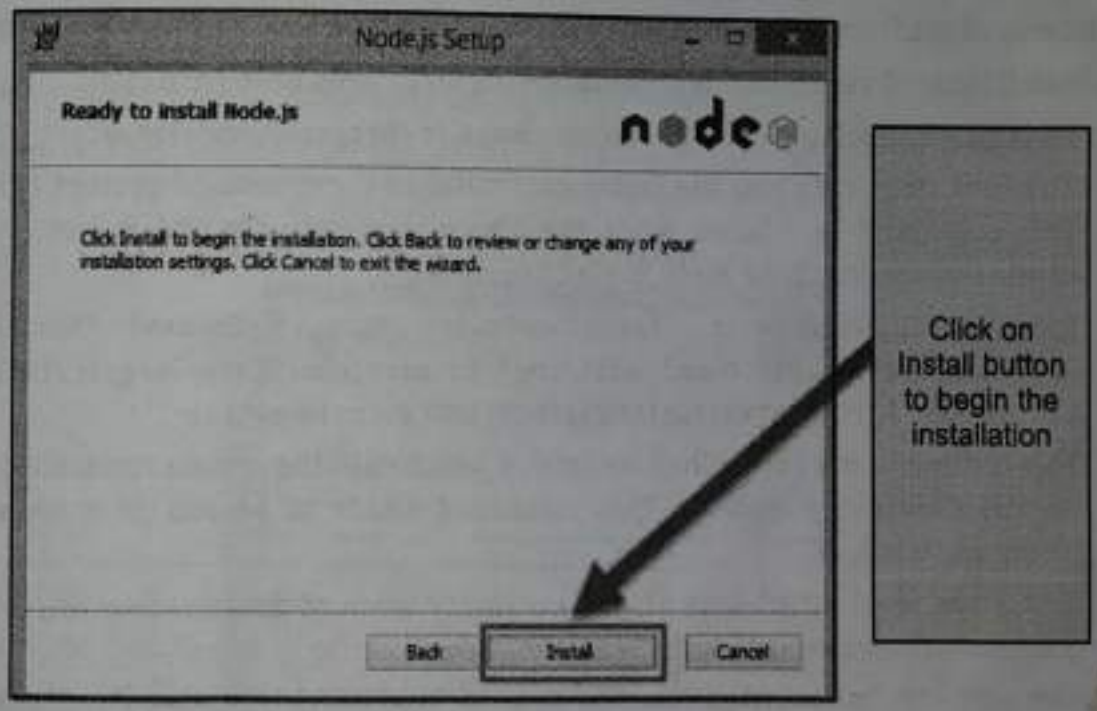


Fig. 1.12

Step 8: Click the Finish button to complete the installation.

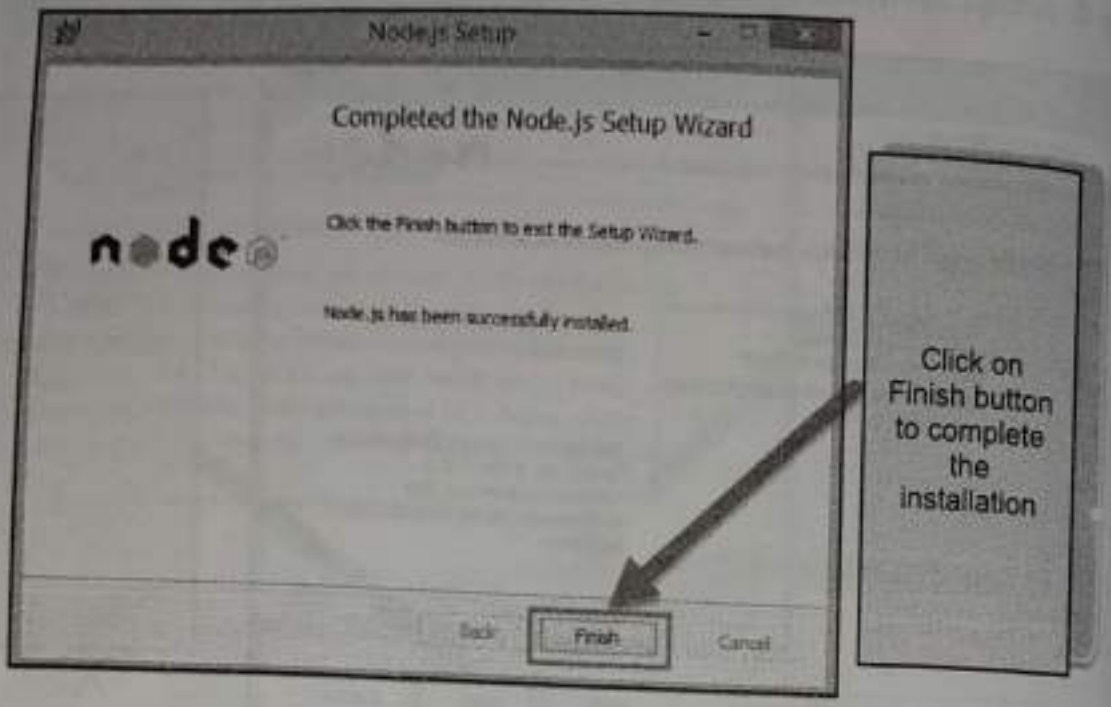


Fig. 1.13

1.8.1 Installing Node through a Package Manager

- The other way to install Node JS on any client machine is to use a "package manager".
- On windows, the node package manager is known as Chocolatey. It was designed to be a decentralized framework for quickly installing applications and tools that you need.
- To install Node JS via Chocolatey, the following steps need to be performed.

Step 1: Installing Chocolatey – The Chocolatey website (`https://chocolatey.org`)

- The first steps is to run the below command in the command prompt windows. This command is taken from the Chocolatey web site and is the standard command for installing Node JS via Chocolatey.

```
@powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((new-
objectwet.webclient).DownloadString('https://chocolatey.org/install.ps1'
))" && SET PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin
```

- This command is a PowerShell command which calls the remote PowerShell script on the Chocolatey website. This command needs to be run in a PowerShell command window.
- This PowerShell script does all the necessary work of downloading the required components and installing them accordingly.

Step 2: The next step is to install Node JS to your local machine using the Chocolatey package manager.

- This can be done by running the below command in the command prompt.

```
cinst nodejs install
```

- If the installation is successful, you will get the message of the successful installation of Node.js.
- Note: If you get an error like "C:\ProgramData\chocolatey\lib\libreoffice\tools\chocolateyInstall.ps1" Then manually create the folder in the path.

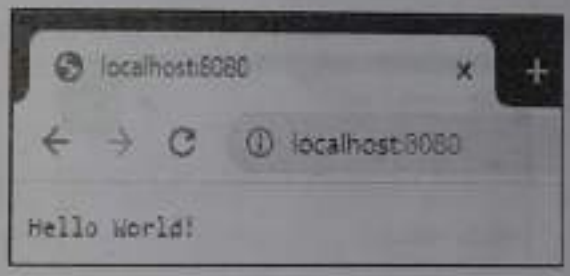
1.9 RUNNING FIRST "HELLO WORLD" APPLICATION IN NODE JS

- Once you have downloaded and installed Node JS on your computer, let's try to display "Hello World" in a web browser.
- Create file Node JS with file name `app.js`
- The most common example "Hello World" of Node JS is a web server.

Program 1.1: Simple "Hello World" program.

```
const http = require('http')
const hostname = '127.0.0.1'
const port = process.env.PORT
const server = http.createServer((req, res) => {
 res.statusCode = 200
 res.setHeader('Content-Type', 'text/plain')
 res.end('Hello World!\n')
})
server.listen(port, hostname, () => {
 console.log(`Server running at http://${hostname}:${port}/`)
})
```

Output:



Explanation:

- The basic functionality of the "require" function is that it reads a JavaScript file, executes the file, and then proceeds to return an object. Using this object, one can then use the various functionalities available in the module called by the require function. So in our case, since we want to use the functionality of http and we are using the require(http) command.

- The createServer() method of http creates a new HTTP server and returns it. The server is set to listen on the specified port and host name. When the server is ready, the callback function is called, in this case informing us that the server is running.
- Whenever a new request is received, the request event is called, providing two objects: a request (an `http.IncomingMessage` object) and a response (an `http.ServerResponse` object).
- These 2 objects are essential to handle the HTTP call. The first provides the request details. In this simple example, this is not used, but you could access the request headers and request data. The second is used to return data to the caller.
- In this case with: `res.statusCode = 200`
- We set the `statusCode` property to 200, to indicate a successful response.
- We set the Content-Type header: `res.setHeader('Content-Type', 'text/plain');` and we close the response, adding the content as an argument to end( ): `res.end('Hello World\n')`

Executing the code:

Step 1: Save the file on your computer: `C:\Users\Your Name\ app.js`

Step 2: In the command prompt, navigate to the folder where the file is stored. Enter the command: `Node app.js`

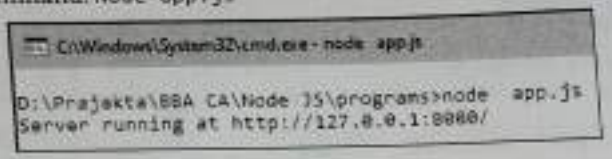


Fig. 1.14

Step 3: Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!

Step 4: Start your internet browser, and type in the address: `http://localhost:8080`

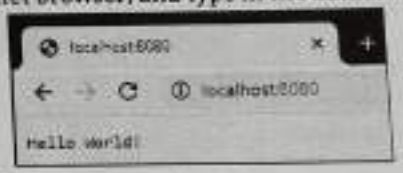


Fig. 1.16

1.10 WORKING IN REPL

- The REPL feature of Node is very useful in experimenting with Node JS code and to debug JavaScript code. It is a quick and easy way to test simple Node.js/JavaScript code.

- REPL stands for Read Eval Print Loop environment.
- REPL is like a Windows console or Unix the system responds with an output in
- To launch the REPL (Node shell), open Mac or UNIX/Linux) and type node as Windows and MAC.

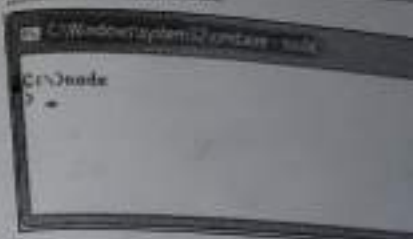


Fig. 1.17.

- Node JS or Node comes bundled wi tasks:
  - **Read:** It is used to reads use structure, and stores in memor
  - **Eval:** It is used to takes and eva
  - **Print:** It is used to prints the re
  - **Loop:** Loops the above comma

Simple Expression:

- Let's try a simple mathematics at

```
> 2 + 7
9
> 2 + (1 * 5) - 4
3
>
```

- Along with adding two integers as in browser's JavaScript.

```
> "Corona" + "Virus"
Corona Virus
```

Variables:

- We can make use variables to s
- To store value we use **var** keyw
- To simply display we don't use

- REPL stands for Read Eval Print Loop and it represents a runtime computing environment.
- REPL is like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode.
- To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux) and type node as shown below. It will change the prompt to > in Windows and MAC.

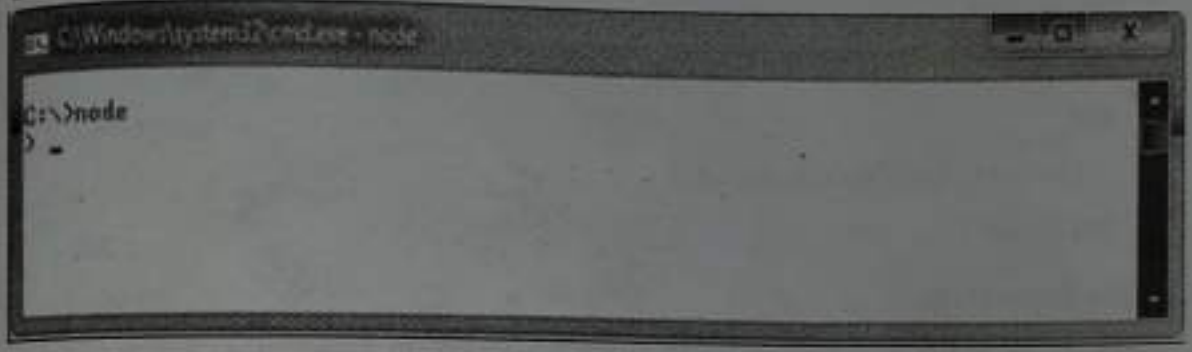


Fig. 1.17: Launch Node JS REPL

- Node JS or Node comes bundled with a REPL environment. It performs the following tasks:
  - **Read:** It is used to reads user's input, parses the input into JavaScript data structure, and stores in memory.
  - **Eval:** It is used to takes and evaluates the data structure.
  - **Print:** It is used to prints the result.
  - **Loop:** Loops the above command until the user presses `ctrl-c` twice.

Simple Expression:

- Let's try a simple mathematics at the Node JS REPL command prompt:

```
> 2 + 7
9
> 2 + (1 * 5) - 4
3
>
```

- Along with adding two integers the + operator is also used to concatenates two strings as in browser's JavaScript.

```
> "Corona" + "Virus"
Corona Virus
```

Variables:

- We can make use variables to store values and print later like any simple program.
- To store value we use **var** keyword.
- To simply display we don't use **var** keyword.

- If **var** keyword is not used, then the value is stored in the variable and printed. Whereas if **var** keyword is used, then the value is stored but not printed. You can print variables using **console.log()**.

```
> a = 100
100
> var b = 200
undefined
> a + b
300
> console.log("Hello World")
Hello World
```

Multiline Expression:

- Node REPL supports multiline expression similar to JavaScript. Following is the example of do-while loop.

```
node
> var x = 0
undefined
> do {
... x++;
... console.log("x: " + x);
... }
while ( x < 5 );
x: 1
x: 2
x: 3
x: 4
x: 5
undefined
>
```

- Comes automatically when you press Enter after the opening bracket. Node automatically checks the continuity of expressions.

Underscore Variable:

- You can use underscore (_) to get the last result:


```
node
> var p = 100
undefined
> var q = 200
undefined
> p + q
300
> var sum = _
undefined
> console.log(sum)
300
undefined
>
```

REPL Commands:

- **.help**: List of all commands.
- **.break**: Exit from multiline expressio
- **.clear**: Exit from multiline expressio
- **.save filename**: Save the current No
- **.load filename**: Load file content in
- **ctrl + c**: Terminate the current comm
- **ctrl + c twice**: Terminate the Node
- **ctrl + d**: Terminate the Node REPL.
- **Up/Down Keys**: See command hist
- **tab Keys**: List of current command

Quitting REPL:

- As mentioned above, you will need

```
node
>
(^C again to quit)
>
```

```
node
> var p = 100
undefined
> var q = 200
undefined
> p + q
300
> var sum = _
undefined
> console.log(sum)
300
undefined
>
```

REPL Commands:

- **.help**: List of all commands.
- **.break**: Exit from multiline expression.
- **.clear**: Exit from multiline expression.
- **.save filename**: Save the current Node REPL session to a file.
- **.load filename**: Load file content in current Node REPL session.
- **ctrl + c:** Terminate the current command.
- **ctrl + c twice**: Terminate the Node REPL.
- **ctrl + d:** Terminate the Node REPL.
- **Up/Down Keys**: See command history and modify previous commands.
- **tab Keys:** List of current commands.

Quitting REPL:

- As mentioned above, you will need to use `ctrl-c` twice to come out of Node JS REPL.

```
node
>
(^C again to quit)
>
```

Summary

- Over the years, most of the applications were based on a stateless request-respons framework. In these sorts of applications, it is up to the developer to ensure the righ code was put in place to ensure the state of web session was maintained while th user was working with the system.
- But with Node JS web applications, you can now work in real-time and have a 2- wa communication. The state is maintained, and the either the client or server can star the communication.
- The Node JS framework is available for a variety of operating systems right from Windows to Ubuntu and OS X. Once the Node JS framework is installed you can star building your first Node JS applications.
- Node JS also has the ability to embedded external functionality or extended functionality by making use of custom modules. These modules have to be installed separately. An example of a module is the MongoDB module which allows you to work with MongoDB databases from your Node JS application.

Check Your Understanding

1. How do you kill a process in Node.js?
   (a) CTRL+A (b) CTRL+K
   (c) CTRL+B (d) CTRL+C
2. Node uses _____ engine in core.
   (a) Chrome V8 (b) Microsoft Chakra
   (c) Node EN (d) Internet Explorer Engine
3. How many Node object methods are available?
   (a) 16 (b) 17
   (c) 18 (d) 19
4. What is the default scope in Node JS application?
   (a) Global to function (b) Local to object
   (c) Local (d) Global
5. Which extension is used to save Node JS files?
   (a) .java (b) .node.
   (c) .txt (d) .js

6. REPL stands for_____.
   (a) Read Eval Print Loop
   (b) Research Eval Program Learn
   (c) Read Earn Point Learn
   (d) Read Eval Point Loop
7. Node JS is _____ Language.
   (a) Server side
   (b) Client side
   (c) Both
   (d) None
8. Command to start Node REPL is____.
   (a) $ node repl
   (b) $ node start
   (c) $ node
   (d) $ node Console
9. Node JS is_____.
   (a) Synchronuos
   (b) Asynchronous

ANSWER KEY

1. (d)	2. (a)	3. (c)	4. (c)	5. (d)
6. (a)	7. (a)	8. (c)	9. (b)	

Practice Questions

Q.I: Answer the following questions in short.

1. What is Node JS?
2. What are the advantages of Node JS?
3. What is Client side scripting?
4. What is Server side scripting?
5. Explain Web server.
6. REPL stands for? — Read Eval print Loop
7. What are applications of Node JS? —
8. NodeJS is a framework. True or False.

Q.II: Answer the following questions.

1. What is difference between Node JS and Agular JS?
2. What is difference between Node JS and Java Script?
3. What is difference between Client Side Scripting and Server Side Scripting?
4. Explain Node JS Process Model?
5. Write steps to install Node JS on Windows through Dashboard?
6. Write steps to install Node JS through Package Manager?

7. Node JS can be use to develop which Applications?
8. Explain REPL in the context of Node.js.
9. What is difference between front end and back end development?
10. Explain asynchronous and synchronous in Node.js.

Q.III: Define the terms.

1. Client
2. Server
3. Web Server
4. Middleware
5. Blocking
6. Non Blocking

2... Node JS Modules

Objectives...

- To learn about Functions.
- To study Node JS Buffer.
- To study Module and its type.
- To understand Module.Exports.

2.1 INTRODUCTION

- In Node JS, a module is a collection of JavaScript functions and objects that can be used by external applications. It is a set of functions which we would like to include in our application.
- Node JS modules are a type of package that can be published to NPM (Node Package Manager).
- Modules in Node JS are a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node JS application.
- In the Node JS module system, each file is treated as a separate module. So, if we are creating a `demo.js` file, this implies we are creating a module in Node. Basically, modules help us encapsulating our code into manageable chunks.
- Anything that we define in our module (i.e., in our JavaScript file) remains limited to that module only, unless we want to expose it to other parts of our code. So, anything we define inside our module remains private to that module only.
- In this chapter, we will be initially dealing with the basics of functions followed by creating and using different modules.

2.2 FUNCTIONS

- A function is a block of statements, which is used to perform a specific task. It is a block of organized, reusable code that is used to perform a single, related action.
- Functions provide better modularity for the application and a high degree of code reusability.

- A function is composed of a sequence of statements called the function body. Values can be passed to a function, and the function will return a value.
- We know that, Node JS is an interpreter and environment for the JavaScript with some specific useful libraries. So, let us see how to write a function in JavaScript.

2.2.1 Defining Functions

- A function definition (also called a function declaration, or function statement) consists of the function keyword, followed by:
  - The name of the function.
  - A list of parameters to the function enclosed in parentheses and separated by commas.
  - The JavaScript statements that define the function, enclosed in curly brackets, {...}.
- A normal function structure in JavaScript is defined as follows:

```
function functionName([param[, param, [..., param]]]){
    // function body
    // optional return;
}
```

- All functions return a value in JavaScript. In the absence of an explicit return statement, a function returns undefined.
- For example, the following code defines a simple function named `calcRectArea`.

Program 2.1: Write a program to calculate area of rectangle using function.

```
function calcRectArea(width, height) {
  return width * height;
}

//invoking the function and printing the result
console.log("Area ::" + calcRectArea(7,3));
```

Output:

```
C:\BCA>node Funcl.js
Area :: 21
```

- Let us see another example, where function is not returning any value. But we know that even though not specified, every function returns the value called `undefined`.

Program 2.2: Program to create a function which is not returned any value.

```
function Display(){
  console.log("Hello World !\n");
}
console.log("Value returned : " + Display());
```

Output:

```
C:\BCA>node Func2.js
Hello world !
Value returned : undefined
```

- Primitive parameters (such as a number) are passed to functions by value. The value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.
- If we pass an object (i.e. a non-primitive value, such as array or a user-defined object) as a parameter and the function changes the object's properties, that change is visible outside the function, as shown in the following program.

Program 2.3: Program to show how to pass an object as a parameter in the function.

```
function ShowData(obj) {
   obj.name='Janardan Pawar';
}
var stud = {roll: 101, name: 'Janardan', percentage:72.45};
var old_name, modified_name;

console.log ("Before Function Call:");
console.log(stud);
old_name = stud.name; // old_name gets the value "Janardan'

ShowData(stud);
Console.log("\n After Function Call:");
Modified_name = stud.name; // modified name gets the value 'Janardan Pawar'

console.log(stud);
```

Output:

```
C:\BCA>node Func3.js
Before Function Call:
{ roll: 101, name: 'Janardan, percentage: 72.45 }
After Function Call:
{ roll: 101, name: 'Janardan Pawar', percentage: 72.45 }
```

2.2.2 Function Expressions (Anonymous Function)

- **Function Expression** allows us to create an anonymous function which doesn't have any function name. This is the main difference between Function Expression and Function Declaration.
- A function expression has to be stored in a variable and can be accessed using *variableName*.
- For example, the function `calcRectArea` could have been defined as:

Program 2.4: Program for anonymous function.

```
const calcRectArea = function(width, height) { return width * height}
var result = calcRectArea(7, 3)
console.log("Value of result = " + result);
```

Output:

```
C:BCA>node Func4.js
Value of result = 21
```

2.2.3 Function Scope

- JavaScript permits us to define a function in other function, which is nested functio Just like any other programming languages, JavaScript has local and global variable
- Variables defined inside a function cannot be accessed from anywhere outside th function, because the variable is defined only in the scope of the function. However, function can access all variables and functions defined inside the scope in which it defined.
- In other words, a function defined in the global scope can access all variables define in the global scope. A function defined inside another function can also access a variables defined in its parent function, and any other variables to which the paren function has access.
- Following program shows accessibility of the variables (local and global).

Program 2.5: Program for local and global variables.

```
// global scope
var roll = 101,
    name = 'Janardan',
    phy = 67,
    che = 72,
    maths = 65;

function totalMarks() {
   return phy+che+maths;
}
console.log("Total Marks Secured = " + totalMarks()); // Returns value 204

// A nested function example
function getScore() {
   var mar = 85,
       sub = 'Marathi';      //local scope

   function showMarks() { //this is a nested function inside the getScore()
```

```
    return name + ' scored ' + mar + ' in the subject ' + sub;
  }
  return showMarks(); //invoking the nested function
}
console.log(getScore()); //invoking the global function
```

Output:

```
C:\BCA>node Func5.js
Total Marks secured = 204
Janardan scored 85 in the subject Marathi
```

Function with Default Arguments:

- Default function parameters allow named parameters to be initialized with default values if no value or undefined is passed.

Program 2.6: Program for function with default arguments.

```
function multiply(num1, num2 = 1) {
return num1 * num2;
}
console.log(multiply(7, 3));
console.log(multiply(8));
```

Output:

```
21
8
```

Solved Programs Based on Functions

Program 2.7: Write a Node JS script to accept radius from the user and calculate area of the Circle.

```
const prompt = require('prompt-sync')();
function calcArea(radius) {
   return 3.14 * radius * radius ;
}
var rad = prompt('Enter radius of the Circle : ');
var result = calcArea(rad);
console.log("Area of the Circle having radius " + rad + " is " + result );
```

Output:

```
C:\BCA>node Area_circle.js
Enter radius of the circle : 3.42
Area of the Circle having radius 3.42 is 36.726696
```

Program 2.8: Write a Node JS script to check a given number is even or odd using the function.

```
const prompt = require('prompt-sync')();
function oddEven(num) {
```

```
    return (num%2==0)? "Even" : "Odd";
}
var n = prompt('Enter a number : ');
console.log("The given number is " + oddEven(n));
```

Output:

```
C:\BCA>node odd_Even.js
Enter a number : 5
The given number is odd

C:\BCA>node odd_Even.js
Enter a number : 26
The given number is Even
```

Program 2.9: Write Node JS program that accepts principle, rate of interest, time and compute the simple interest.

```
const prompt = require('prompt-sync')();
function calSimpleInterest(p,n,r) {
    var SI = (p*n*r)/100;
    return SI;
}
const p_amt = prompt('Enter Principle Amount : ');
const roi = prompt('Enter Rate of Interest :');
const time = prompt('Enter duration/time :');
var answer = calSimpleInterest(p_amt,roi,time);
console.log("Simple Interest = " + answer);
```

Output:

```
C:\BCA>node simple_interest.js
Enter Principle Amount : 10000
Enter Rate of Interest :10
Enter duration/time :12
Simple Interest = 12000
```

Program 2.10: Write a Node JS script to check a given number is perfect or not using function.

```
const prompt = require('prompt-sync')();
function isPerfectNumber(num) {
    var sum = 0;
    for(var i=1;i<=num/2;i++){
        if(num % i === 0)
            sum += i;
    }
    if(sum == num && sum != 0)
```

```
        console.log("The given number " + num + " is a perfect number.");
    else
        console.log("The given number " + num + " is Not a perfect number.");
}
var n = prompt('Enter a number : ');
isPerfectNumber(n);
```

Output:

```
C:\BCA>node Perfect_Number.js
Enter a number : 28
The given number 28 is a perfect number.

C:\BCA>node Perfect_Number.js
Enter a number : 35
The given number 35 is Not a perfect number.
```

Program 2.11: Write a Node JS program to calculate factorial of given number using function.

```
const prompt = require('prompt-sync')();
function factorial(n) {
    var answer = i;
    if(n<0)
        return ' not possible(Negative number)';
    else if (n == 0 || n == 1)
        return answer;
    else {
        for(var i = n; i >= 1; i--) {
            answer = answer * i;
        }
        return answer;
    }
}
const n = prompt('Enter a number : ');
answer = factorial(n)
console.log("The factorial of " + n + " is " + answer);
```

Output:

```
C:\BCA>node Factorial.js
Enter a number : 5
The factorial of 5 is 120

C:\BCA>node Factorial.js
Enter a number : 0
```

```
The factorial of 0 is 1

C:\BCA>node Factorial.js
Enter a number : -2
The factorial of -2 is not possible(Negative number)
```

2.3 BUFFER

- Buffer is an object property on Node's global object, which is heavily used in Node to deal with streams of binary data. As it is globally available, there is no need to require it in our code.
- Buffer is actually a chunk of memory allocated outside of the V8 heap. V8 is the default JavaScript engine which powers Node and Google Chrome. In Node, buffers are implemented using a JavaScript typedArray (Uint8Array), but that does not mean the memory allocated to buffer is inside of the V8 heap. It is still explicitly allocated outside the V8 heap.
- So, we can think of buffer as some kind of array which is a lower level data structure to represent a sequence of binary data, but there is one major difference: Unlike arrays, once a buffer is allocated, it cannot be resized.

2.3.1 Why we use Buffer?

- Pure JavaScript works well with Unicode-encoded strings. In fact, it is the Unicode in your browser that states that 76 should represent L. Pure JavaScript do not handle straight binary data very well; this is fine in browsers where most data is in the form of strings. However, Node JS have to face with reading and writing to the file system and TCP streams thus it makes it necessary to deal with purely binary streams of data.
- Node has a way to handle binary data, Buffer class is the primary data structure in a node and used with most I/O operations. In node, each buffer represents to some raw memory allocated outside V8. A buffer acts like an array of integers, once allocated it cannot be resized.
- Buffers act somewhat like arrays of integers, but are not resizable and have a whole bunch of methods specifically for binary data. The "integers" in buffer represents a byte and they are limited to values from 0 to 255 (2^8-1).

2.3.2 How to create Buffers?

- Below mentioned methods of Buffer can be used to create buffers.

1. `Buffer.from()`:This method is used to generate a buffer from a string, object, array or buffer.

   Syntax: `Buffer.from(obj, encoding);`

   Where,

`Obj`: Object by which should be filled in, Different types that we can fill are (String,Array,Buffer,arrayBuffer)

`Encoding`: The encoding of string. Default value is utf8. This is an optional parameter.

2. `Buffer.alloc()`: It takes a size (integer) as an argument and returns a new initialized buffer of the specified size (i.e., it creates a filled buffer of a certain size).

Syntax: `Buffer.alloc(size, fill, encoding);`

Where,

`size`: Desired length of new Buffer. It accepts integer type of data.

`fill`: The value to prefill the buffer. The default value is 0.It accepts any of the following: integer, string, buffer type of data.

`encoding`: It is Optional. If buffer values are string , default encoding type is utf8.

3. `Buffer.allocUnsafe()`: This method creates new buffer object of the specified size but it will not initialize the values thus it is the no-prefill buffer. The segment of allocated memory is uninitialized; an allocated segment of memory might contain sensitive or confidential data from older buffers.

Syntax: `Buffer.allocUnsafe(size);`

Where, `Size` : Desired length of new Buffer. It accepts integer type of data.

Program 2.12: Program creates a zero-filled buffer of the specified length.

```
var MyBuffer1 = Buffer.alloc(15);
console.log(MyBuffer1);
```

Output:

```
<Buffer 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 >
```

- Here we have allocated the memory of 20 bytes or size of 20 bytes to buffer object. Here we have not specified the fill value thus by default it will take zero value in hexadecimal format.

Program 2.13: Program creates buffer from a given array.

```
var MyBuffer2 = Buffer.from([ 8, 6, 0, 5, 4, 12, 9, 6]);
console.log(MyBuffer2);
```

Output:

```
<Buffer 08 06 00 05 04 0c 09 06>
```

- Here, this initializes the buffer to the contents of this array. Keep in mind that the contents of the array are integers representing the bytes.
- See following code, will initialize the buffer to a binary encoding of the first string as specified by the second argument (in this case, 'utf-8'). 'utf-8' is by far the most common encoding used with Node JS.

Program 2.14: Program to initialize the buffer to a binary encoding.

```
var MyBuffer3 = Buffer.from("Indira College", "utf-8");
console.log(MyBuffer3);
```

Output:

```
<Buffer 49 6e 64 69 72 61 20 43 6f 6c 6c 65 67 65>
```

2.3.3 Writing to Buffers

- Till now, we have seen few ways to create the buffer in Node JS. Now, let us see how to write on to the buffers.
- **Syntax:**

```
BufferName.write(string, offset, length, encoding)
```

Where,

String: This is the string data to be written to buffer.

Offset: This is the index of the buffer to start writing at. Default value is 0.

Length: This is the number of bytes to write. Defaults to buffer.length.

Encoding: Encoding to use. 'utf8' is the default encoding.

Return Value

- This write() method returns the number of octets written. If there is not enough space in the buffer to fit the entire string, it will write a part of the string.
- Let us explore it a bit using Command Prompt of Node JS.

```
C:\BCA>node
Welcome to Node.js v14.15.1.
Type ".help" for more information.
> var MyBuffer = Buffer.alloc(20);
undefined
> MyBuffer.write("Hello World!",'utf-8');
12
```

- Here, we have a created a buffer of 20 bytes. MyBuffer.write() has returned the value 12. This means that we wrote to 12 bytes of data of the buffer.
- Now, let us consider following snippet. When `MyBuffer.write()` has 3 arguments, the second argument indicates an offset, or the index of the buffer to start writing at.

```
MyBuffer.write("Indira",6,'utf-8');
6
```

2.3.4 Reading from Buffers

- Probably the most common way to read buffers is to use the `toString()` method, since many buffers contain text.

- **Syntax**: `buf.toString(encoding, start, end)`

Where,

Encoding: This is encoding type to be used. The 'utf8' is the default encoding type.

Start: This denotes the beginning index to start reading. The defaults value is 0.

End: This denotes the end index to end reading. The default value is the complete buffer.

- Let us have a look at following snippet.

```
> MyBuffer.toString('utf-8');
'Hello Indira\x00\x00\x00\x00\x00\x00\x00\x00'
```

- Again, the first argument is the encoding. In this case, it can be seen that not the entire buffer was used.
- As we know, how many bytes we've written to the buffer, we can simply add more arguments to "stringify" the slice. Following code will read the contents of the buffer named `MyBuffer` from index 0 to 12.

```
> myBuffer.toString('utf-8, 0, 12);
'Hello Indira'
```

- There are so many methods and properties which belong to buffer. Few of them are used in the code given below.
- Let us see, entire command prompt of the above mentioned methods and properties of the buffer.

```
Type ".help" for more information.
> var MyBuffer = Buffer.alloc(20);
undefined
> MyBuffer.write("Hello World!",'utf-8');
12
> MyBuffer.write("Indira",6,'utf-8');
6
> MyBuffer.toString('utf-8');
'Hello Indira\x00\x00\x00\x00\x00\x00\x00\x00'
> MyBuffer.toString('utf-8',0, 12);
'Hello Indira'
> Buffer.isBuffer(MyBuffer);
true
> MyBuffer.length
20
> MyBuffer.slice(0,5);
<Buffer 48 65 6c 6c 6f>
> const data = MyBuffer.slice(0,5);
undefined
> data.toString()
'Hello'
>
```

2.3.5 Concatenate Buffers

- Following Syntax is used to concatenate Node buffers to a single Node Buffer.

Syntax : Buffer.concat(list, Length)

Where,

List: This is the array List of the Buffer objects which are to be concatenated.

Length: Here it denotes the total length of the buffers when they are concatenated, it is optional.

Program 2.15: Program for concatenation of buffers.

```
var buffer1 =  Buffer.from('Hi ');
var buffer2 =  Buffer.from('Nirali Publication');
var buffer3 = Buffer.concat([buffer1,buffer2]);
console.log("buffer3 content are: " + buffer3.toString());
```

Output: buffer3 content are: Hi Nirali Publication

2.3.6 Compare Buffers

- The compare() method compares two buffer objects and it returns a number according to their differences
- Following Syntax is used to compare two node Buffers

Syntax : buf.compare(buffer1,buffer2) ; buffer1.compare(buffer2)

- **Note:** This method compares two buffer objects and returns a number defining their differences:
  - Shows 0 if they are equal
  - Shows 1 if buffer1 is higher than buffer2
  - Shows -1 if buffer1 is lower than buffer2

Program 2.16: Program for comparing buffers.

```
var buffer1 = Buffer.from('Nirali');
var buffer2 = Buffer.from('Publication Pune');
var result = buffer1.compare(buffer2);

console.log("Value returned is : " + result);
if(result < 0) {
  console.log(buffer1 +" lower than " + buffer2);
}
else if(result === 0){
  console.log(buffer1 +" is same as " + buffer2);
}
```

```
else {
  console.log(buffer1 +" higher than " + buffer2);
}
```

Output:

```
Value returned is : -1
Nirali lower than Publication Pune
```

2.4 MODULE

- In Node JS, a module is a collection of JavaScript functions and objects that can be used by external applications. It is a set of functions which we would like to include in our application.
- Module in Node JS is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node JS application.
- Collection of files can be considered as a module if its functions and data are made usable to external programs.
- In Node, the modularity is a first class concept. In the Node JS module system, each file is treated as a separate module.
- So, if you are creating, let's say, a `demo.js` file, this implies you are creating a module in Node. Basically modules help us encapsulating our code into manageable chunks.
- Anything that we define in our module (i.e. in our JavaScript file) remains limited to that module only, unless we want to expose it to other parts of our code.
- So, anything we define inside our module remains private to that module only.

2.5 MODULE TYPES

- Node JS includes three types of modules:

1. Core Modules, 2. Local Modules, 3. Third Party Modules

2.5.1 Core Modules

- Core Modules in Node JS are also called as built-in modules which we can use without any further installation. The core modules include basic minimum functionalities of Node JS. These core modules are compiled into its binary distribution and load automatically when Node JS process starts.
- Below mentioned are the few Core Modules in Node JS:

Module Name	Description
`http`	This module includes classes, methods and events to create Node JS http server.

Contd..

path	This module includes methods to deal with file paths.
util	This module includes utility functions useful for programmers.
fs	This module includes classes, methods, and events to work with file I/O.
timers	This module is used to execute a function after a given number of milliseconds.
zlib	This module is used to compress or decompress files.
url	This module includes methods for URL resolution and parsing.
assert	It is used to Provides a set of assertion tests.
buffer	It is used to handle binary data.
timers	It is used to execute a function after a given number of milliseconds.

How to load core modules in Node JS Application?

- We need to import the core module first in order to use it in our application. If we unable to load any module, then we cannot use functionality of the modules. Means, we cannot access the objects and functions available in the modules.
- To include the required module in our application is very easy. We just need to make use of `require()` function as:

  Syntax:

```
var variable_name = require('Module_Name');
```

- We can also use **const** instead of **var** here.
- **Return value:** The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.
- **For Example:**

```
const absolutePath = require('path');
```

- The path module provides a lot of very useful functionality to access and interact with the file system. There is no need to install it. Being part of the Node JS core, it can be used by simply requiring it.

Methods available in the path module:

1. **basename():** It return the last portion of a path. A second parameter can filter out the file extension.

Program 2.17: Program for basename() method.

```
const absolutePath = require('path');

const fileNamel = absolutePath.basename('D:/Study_Data/NodeJS/Notes.docx');
console.log(fileNamel);

const fileName2=
absolutePath.basenase('D:/Study_Data/NodeJS/Notes.docx','.docx');
console.log(fileName2);
```

Output:

```
C:\BCA>node ModuleDemo.js
Notes.docx
Notes
```

- If you are the Windows OS user, then you can use the path as given below.

```
'D:\\Study_Data\\Node JS\\Notes.docx'
```

2. **dirname():** It returns the directory part of the path.

Program 2.18: Program for dirname() method.

```
const absolutePath = require('path');

const dir = absolutePath.dirname('D:/Study_Data/Node JS/Notes.docx');
console.log(dir);
```

Output:

```
C:\BCA>node ModuleDemo2.js
D:/Study_Data/Node JS
```

3. **extname():** It returns the extension part of a path.

Program 2.19: Program for extname() method.

```
const absolutePath = require('path');

const ext = absolutePath.extname('D:/Study_Data/Node JS/Notes.docx');
console.log("File Extension : " + ext);
```

Output:

```
C:\BCA>node ModuleDemo3.js
File Extension: .docx
```

4. **isAbsolute()**: It returns true if it's an absolute path.

Program 2.20: Program for isAbsolute() method.

```
const absolutePath = require('path');

const ans1 = absolutePath.isAbsolute('D:/Study_Data/Node JS/Notes.docx');
console.log("Answer : " ans1);

const ans2 = absolutePath.isAbsolute('Study_Data/Node JS/Notes.docx');
console.log("Answer : " ans2);
```

Output:

```
C:\BCA>node MuduleDemo4.js
Answer : true
Answer : false
```

- Other methods which are available under path module are join(), normalize(), parse(), relative(), etc.

2.5.2 Local Modules

- Local modules are the custom modules defined and developed by the user i.e. coder. Hence, it is also called as User Defined Module. These modules are created locally in our Node JS application. These modules include different functionalities of our application in separate files and folders.

Writing Local Module:

- Creating our own customized module is very easy and consisting of following simple steps.
- Let us write simple module which perform arithmetic operations like addition, subtraction, multiplication and division of any two numbers.
- In Node JS, module should be placed in a separate JavaScript file. So, first, create a file and write the following code in it. Save this file with the name **userDefinedModule.js.**

```
//.js file containing functions which are performing arithmetic operations

// Returns addition of two numbers
exports.add = function (num1, num2) {
   return num1+num2;
};

// Returns difference of two numbers
```

```
exports.subtract = function (num1, num2) {
   return num1-num2;

};

// Returns multiplication of two numbers
exports.multiply = function (num1, num2) {
   return num1*num2;

};

// Returns division of two numbers
exports.divide = function (num1, num2) {
   return num1/num2;
};
```

- Note that, the functions we add to the custom module must be defined with "exports" in front of the function name.
- The "exports" keyword is used to ensure that the functionality defined in this file can actually be accessed by other files.

Loading Local Module:

- Here, we will actually see how to create the application which will call our custom module.
- To use local modules in our application, we need to load it using require() function in the same way as core module. However, we need to specify the path of JavaScript file of the module.

MainApplication.js

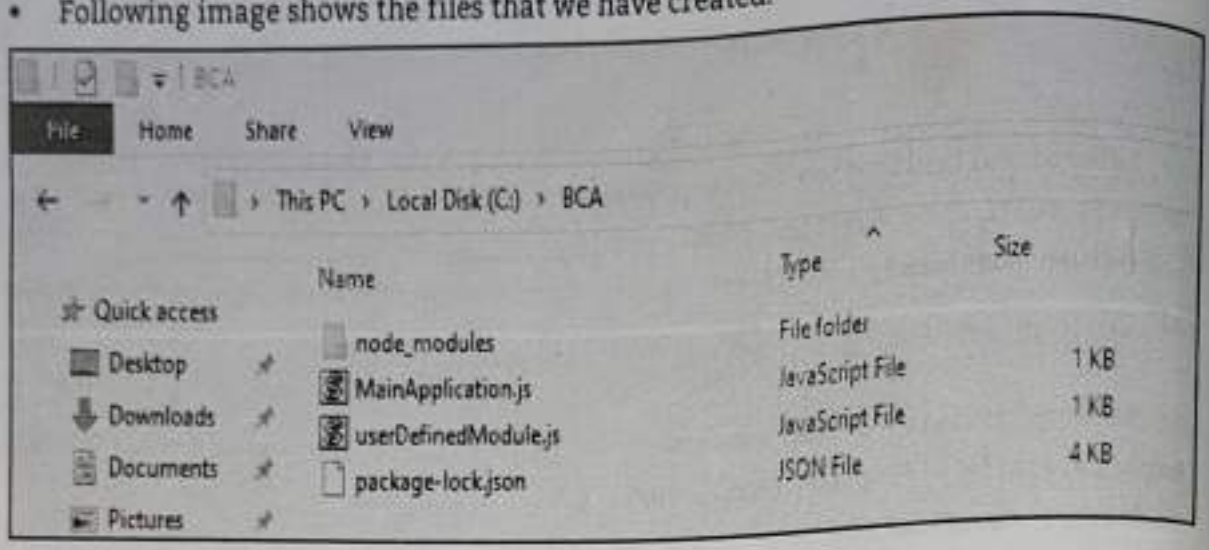
```
//Accessing user defined (custom module) named userDefinedModule

var op = require('./userDefinedModule');

var x=25, y=5;

console.log("Arithmetic Operations on ", x , "and ", y , "are :\n");
console.log("Addition : "+ op.add(x,y));
console.log("Subtraction : "+ op.subtract(x,y));
console.log("Multiplication : "+ op.multiply(x,y));
console.log("Division : "+ op.divide(x,y));
```

- Following image shows the files that we have created.



- When we execute our main application, we will get following **output:**

```
C:\BCA>node MainApplication.js
Arithmetic Operations on 25 and 5 are :
Addition : 30
Subtraction : 20
Multiplication : 125
Division : 5
```

2.6 MODULE.EXPORTS

- Now, we will see how to expose different types as a module using `module.exports`.
- The `module.exports` is a special object which is included in every JavaScript file in the Node JS application by default. The `module` is a variable that represents the current module, and `exports` is an object that will be exposed as a module. So, whatever we assign to `module.exports` will be exposed as a module.

Export Literals:

- `exports` is an object. So it exposes whatever we assigned to it as a module.
- For example, if we assign a string literal then it will expose that string literal as a module.
- Let us see this using simple program.

Program 2.21: Program for export literals.

Greetings.js

```
module.exports = 'Welcome to Indira College of Commerce & Science';
```

mainApp.js

```
var message = require('./Greetings.js');
console.log(message);
```

Output:

```
C:\BCA>node mainApp.js
Welcome to Indira College of Commerce & science
```

- In above program, we add simple and one line code in the file `Greetings.js`. After that import this module in the file named `mainApp.js` and execute it to see the output.

Export Object:

- The export is an object. So, you can attach properties or methods to it.

Program 2.22: Program for export object.

Notices.js

```
exports.notice1 = 'Beware of COVID19.';
//we can also write it as following
module.exports.notice2 = 'Take Necessary Precautionary Measures.';
```

mainApp.js

```
const message = require('./Notices.js');
console.log(message.notice1);
console.log(message.notice2);
```

Output:

```
C:\BCA>node mainApp.js
Beware of COVID19.
Take Necessary Precautionary Measures.
```

- In above program, exposes an object with a string property in `Notices.js` file. Then import the module `Notices` in `mainApp.js` file and execute it to see the output.

Exposing an object with function using exports:

- Example given below exposes an object with the myFun function as a module.
- Import the module `ObjectFunction` in `mainApp.js` file and execute it to see the output.

Program 2.23: Program for exposing an object with function using exports.

ObjectFunction.js

```
module.exports.myFun = function (greet) {
    console.log(greet);
};
```

mainApp.js

```
var data = require('./ObjectFunction.js');
data.myFun('Welcome to Indira College of Commerce & Science, Pune(MH)');
```

Output:

```
C:\BCA>node mainApp.js
Welcome to Indira College of Commerce & Science, Pune(MH)
```

Summary

- Functions are first class citizens in Node's JavaScript, similar to the browser's JavaScript. A function can have attributes and properties also. It can be treated like a class in JavaScript.
- Buffers are instances of the Buffer class in Node.js. Buffers are designed to handle binary raw data. Buffers allocate raw memory outside the V8 heap. Buffer class is a global class so it can be used without importing the Buffer module in an application.
- In Node.js, Modules are the blocks of encapsulated code that communicates with an external application on the basis of their related functionality. Modules can be a single file or a collection of multiples files/folders.
- The reason programmers are heavily reliant on modules is because of their re. usability 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 comes with Node.js installation. These modules can be loaded into the program by using the require function.
- **Local Modules:** Unlike built-in and external modules, local modules are created locally in your Node.js application.
- **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.

Check Your Understanding

1. Which module is used to serve static resources in Node JS?
   (a) static (b) node-resource
   (c) http (d) node-static
2. How Node JS modules are available externally?
   (a) module.spread (b) module.exports
   (c) module.expose (d) None of the above
3. Which of the following module is required for path specific operations?
   (a) Os module (b) Path module
   (c) Fs module (d) All of above

4. Simple or complex functionality organized in a single or multiple JavaScript files which can be reused throughout your Node JS application is called _____.
   (a) Library (b) Package
   (c) Function (d) Module
5. Which is/are the core module(s) in Node.js?
   (a) http (b) url
   (c) path (d) all of the above
6. Which function is used to include modules in Node Js.
   (a) Include() (b) Require()
   (c) Attach() (d) None of the above
7. Which of following is not built-in node module?
   (a) Zlib (b) https
   (c) dgram (d) fsread
8. What does the fs module stand for?
   (a) File Service (b) File System
   (c) File Store (d) None of the above

ANSWER KEY

1. (d)	2. (b)	3. (b)	4. (d)	5. (d)
6. (b)	7. (d)	8. (b)		

Practice Questions

Q.I: Answer the following questions in short.

1. What is function in node JS?
2. Define Anonymous function?
3. List out different core module? — http, path, assert, buffer, timers
4. Write different types of Node JS Modules? — core module, local module, Third-party module
5. Which module is used for file based operations?
6. Define Scope of functions?
7. Write syntax to create Buffer? — BufferName.write()
8. How to read and write buffer, Write Syntax? — toString()

Q.II: Answer the following questions.

1. Explain Modules in Node JS.
2. Explain module.exports in Node JS?
3. What are some of the most popular modules of Node.js?

4. What is function? Write syntax to define function.
5. Write Steps to load Core modules in Node JS?
6. Write Steps to load Local modules in Node JS?

Q.III: Define the terms.

1. Modules
2. Function
3. Core Module
4. Local Module
5. Module.export
6. Buffer

3...

Node Package Manager

Objectives...

- To learn about Node Package Manager.
- To learn how to install packages locally.
- To study how to add dependency in package.json.
- To learn how to install packages globally.
- To learn about updating packages.

3.1 INTRODUCTION

- As we understood in previous chapters that, Node JS is open source server environment which uses JavaScript on the server and available absolutely free.
- You can generate dynamic page contents, can create, open, rewrite, and delete files on the server.
- You can collect form data as well as can add, delete, modify data on database. This all about basic of Node JS to start with NPM (Node Package Manager).
- A package contains all the files needed for a module and modules are the JavaScript libraries that can be included in Node project according to the requirement of the project.

3.2 WHAT IS NPM?

- NPM stands for Node Package Manager. It is written entirely in JavaScript and was developed by Isaac Z. Schlueter in 2010. NPM is the world's largest software registry. Many organizations use npm to manage private development as well.
- It is a package manager for Node JS packages and modules present in an application. So it is basically responsible for managing all your Node JS Packages.

- NPM comes bundled with Node JS. So when you install Node JS, NMP also gets installed and it happens after version 0.6.3 of Node.js. That's why it is a default Package Manager of Node JS.
- NPM is free to use. You can download all npm public software packages without any registration or logon.
- All npm packages are defined in files called **package.json**. The content of Package.jsonmust be written in JSON.
- NPM consists of three distinct components:
  1. **Website:** Use the website to discover packages, set up profiles, and manage other aspects of your npm experience.
  2. **Command Line Interface (CLI):** NPM includes a CLI that can be used to download and install software: C:\>npm install <package>
  3. **Registry:** The registry is a large public database of JavaScript software and the meta-information surrounding it.
- An excess of Node JS libraries and applications are published on npm, and many more are added every day. These applications can be searched for on `http://npmjs.org/`. Once you have a package you want to install, it can be installed with a single command line command.
- NPM is very simple to use: you only have to run `npm install async`, and the specified module will be installed in the current directory under ./node_modules/. Once installed to your node_modules folder, you'll be able to use require() on them just like they were built-ins.
- Node Package Manager provides two main functionalities:
  1. It Provides Online repository for Packages/Modules for Node JS which you can easily search online on their official site which is `search.nodejs.org`.
  2. It provides a command line utility to install Node JS Packages. It allows you to do version management and also the dependency management of Node JS packages.

3.2.1 Need for NPM

1. It helps in incorporating the pre-built packages into our project.
2. It assists in downloading various standalone tools which can be used right away.
3. Using NPM, you can even run packages without having to download it.
4. Developers often use NPM to share their code with other NPM users.
5. Helps in restricting the code to specific developers and forming virtual teams using orgs.

6. Helps in managing and maintaining various versions of codes and their dependencies.
7. NPM also updates the application with the update in underlying codes.

3.2.2 Advantages of NPM

1. NPM is free to use. As it is open source package system.
2. It is simpler than SOAP.
3. It is a default package manager for Node JS.
4. Completely open source and written in JavaScript
5. As it manages all the packages and modules for Node JS and it consists of also command line client NPM, so it's become world's largest software registry.

3.3 PACKAGE

- Package contains all the files which are needed for modules and modules are the JavaScript libraries that can be included in a Node JS project as per the requirements of the project.
- The npm registry contains packages, A package is a file or directory that is described by a Package.json file. A package must contain a Package.json file in order to be published to the npm registry.

What is the Package.json file?

- The Package.json file is one of the most important files when working with Node applications. With this single file, we keep track of all of our external dependencies. This file is also used to tell others about our project. The "package.json" file is used to hold the metadata about a particular project. This information provides the node package manager the necessary information to understand how the project should be handled along with its dependencies.
- The Package.json file contains information such as the project description, the version of the project in a particular distribution, license information, and configuration data.
- The Package.json file is usually located at the root directory of a Node JS project.
- Let's take an example of how the structure of a module looks when it is installed via npm. The below snapshot shows the file contents of the express module when it is included in your Node JS project. From the snapshot, you can see the Package.json file in the express folder.

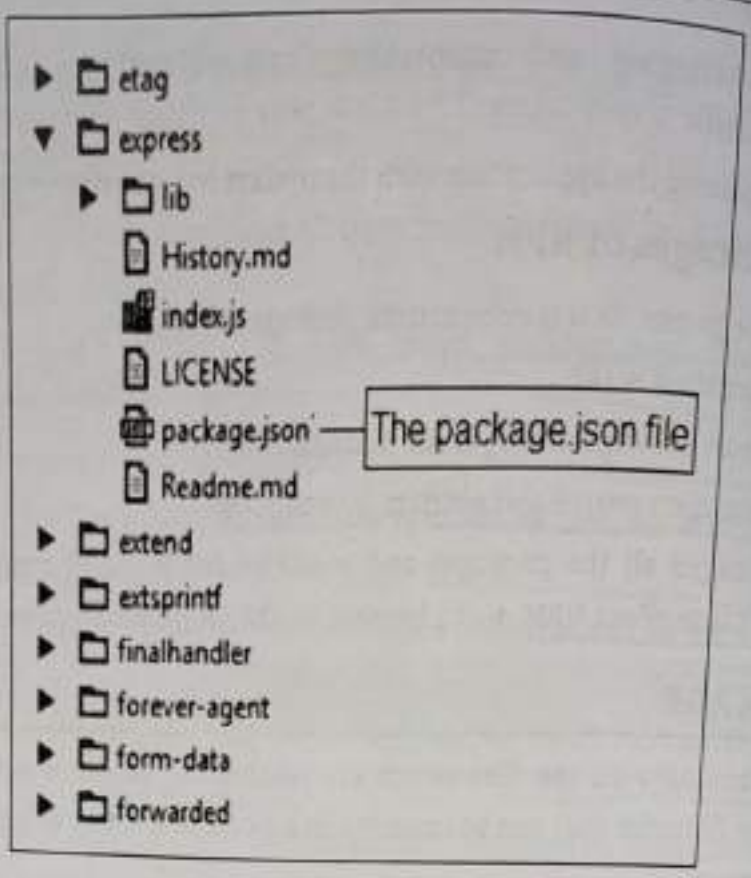


Fig. 3.1

- If you open the Package.json file you will see a lot of information in the file.
- See following figure, the express@~4.13.1 mentions the version number of the express module being used.

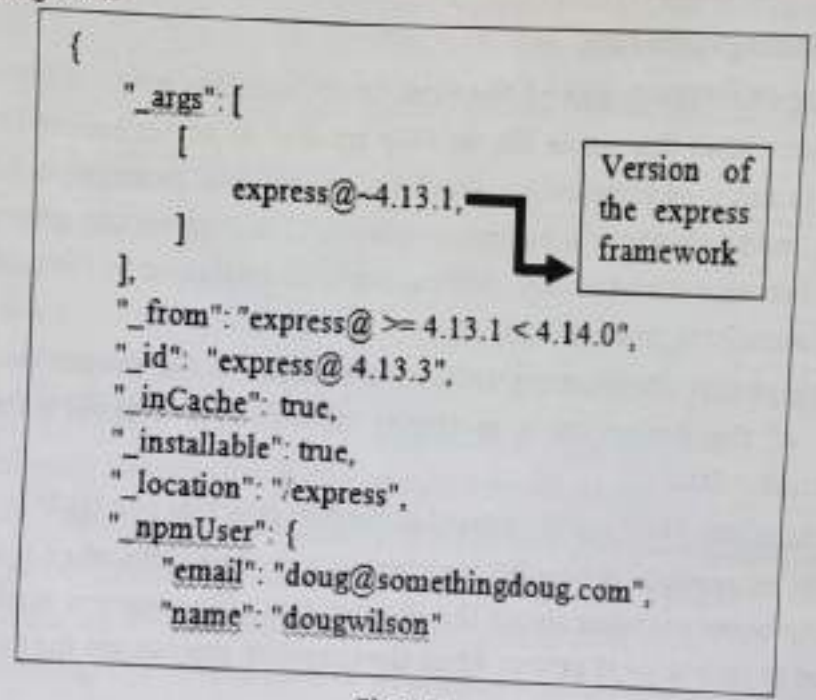


Fig. 3.2

3.4 INSTALLING PACKAGES LOCALLY

- Now, we will look how to install packages. There are two ways of installation the packages, either locally or globally. So the question arises when should we install globally and when should we install locally. You chose the method which based on how actually you want to use those packages.
- You should install packages locally when you want to depend on the package from your own module, using something such as Node JS require. This is npm install's default behavior.
- When you want to use a package as a command line tool, (like grunt CLI), then you have to install it globally.

3.4.1 Installing a Package

- You can download a package with the command: `npm install <package_name>`

```
C:\WINDOWS\system32>npm install express
npm WARN saveError ENOENT: no such file or directory, open 'C:\WINDOWS\system3
2\package.json'
npm [illegible] created a lockfile as package-lock.json. You should commit this fil
e.
npm WARN enoent ENOENT: no such file or directory, open 'C:\WINDOWS\system32\p
ackage.json'
npm WARN system32 No description
npm WARN system32 No repository field.
npm WARN system32 No README data
npm WARN system32 No license field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 3.703s
found 0 vulnerabilities
```

Fig. 3.3

- This creates the node_modules directory in your current directory (if one doesn't exist yet) and downloads the package to that directory.
- If you want to confirm that npm install worked correctly, you should check to see that a node_modules directory exists and that it has a directory for the package(s) that you have installed.
- For example, Take for instance that you install a package called express, you can confirm that it worked correctly by checking that a node_modules directory now exists and that the directory has a subdirectory named express.

For Microsoft Windows:

```
npm install express
C:\ dir node_modules
#=> express
```

- If a Package.json file does not exist in the local directory, the latest version of the package will be installed.
- However, if a Package.json file exists, npm will install the latest version that satisfies the semver (semantic versioning) rule that is declared in the package.json.

3.4.2 Using the Installed Package in Your Code

- Once you have run `npm install <package name>` and the package is in the node_modules directory, it can be used in your code. For instance, when you are creating a Node JS module, you can use require to access it.
- **Example:** Create a file and name it index.js, and add the following code:

```
// index.js
var express = require('express');
app.use(express)
console.log("successfully required a package")
```

When you run the code, you should get the `output: successfully required a package`.

- If you had not installed express properly, you will get the following error message:

```
module.js:340
throw err;
      ^
Error: Cannot find module 'express'
```

- To fix this error, you should run npm install express in the same directory as your index.js

3.4.3 How to Uninstall Local Packages

- If you want to remove a package directly from your node_modules directory, you should use:

```
npm uninstall <package>
npm uninstall lodash
```

- If you want to remove it from the dependencies in package.json, you have to use the save flag:

```
npm uninstall --save lodash
```

Note: if you had installed the package as a "devDependency" (i.e. with `--save-dev`) then `--save` will not remove it from package.json. You will need to use `--save-dev` to uninstall it.

- If you want to confirm that npm uninstall worked correctly, you should find the node_modules directory. Ensure that it no longer contains a directory for the package(s) you uninstalled.
- This can be done by running:
  - `dir node_modules` on Windows.
- **For example,** Install a package called axios. Then confirm that it run successfully by listing the contents of the node_modules directory and seeing a directory called axios.
- Uninstall axios with npm uninstall. Then confirm that it run successfully by listing the contents of the node_modules directory and confirming the absence of a directory called axios.

Install Axios:

```
npm install axios
dir node_modules          # use `ls node_modules` for Unix
```

Uninstall Axios:

```
#=> axios
> npm uninstall axios
> dir node_modules        # use `ls node_modules` for Unix
#=>
```

3.5 ADDING DEPENDENCY IN PACKAGE.JSON

- We can add dependencies to a `Package.json` in two ways:
  1. File from the Command Line.
  2. Manually Editing the Package.json File.

Adding Dependencies to a `Package.json` File from the Command Line:

- To add dependencies and devDependencies to a `Package.json` file from the command line, we can install them in the root directory of package using the `--save-prod` flag for dependencies (the default behavior of npm install) or the `--save-dev` flag for devDependencies.
- To add an entry to the "dependencies" attribute of a `Package.json` file, on the command line, following command will be used:

```
npm install <package-name> [--save-prod]
```

- To add an entry to the "devDependencies" attribute of a `Package.json` file, on the command line, run the following command:

```
npm install <package-name> --save-dev
```

Manually Editing the Package.json File:

- To add dependencies to a Package.json file, in a text editor, add an attribute called "dependencies" that references the name and semantic version of each dependency:

```
{
  "name": "my_package",
  "version": "1.0.0",
  "dependencies": {
    "my_dep": "^1.0.0",
    "another_dep": "~2.2.0"
  }
}
```

- To add devDependencies to a Package.json file, in a text editor, add an attribute called "devDependencies" that references the name and semantic version of each devDependency:

```
"name": "my_package",
"version": "1.0.0",
"dependencies": {
  "my_dep": "^1.0.0",
  "another_dep": "~2.2.0"
},
"devDependencies" : {
  "my_test_framework": "^3.1.0",
  "another_dev_dep": "1.0.0 - 1.2.0"
}
```

3.6 INSTALLING PACKAGES GLOBALLY

- In previous section, we talked about working with Package.json, Now, will look on how to install, update and uninstall packages globally. As we have discussed earlier that, there are two options available to install a package: it is either install it locally or install it globally. The choice on which kind of installation is dependent on how you want to use the package.
- Whenever you want to use a package as a command line tool, you should install it globally. In this way, it will work no matter which directory is current. This is the choice you should use if you were installing grunt.

- Whereas, when you want to depend on the package from your own module, you should install it locally. This is the choice you would normally use if you are using require statements.
- You can use the command npm install -g <package>, for install:

```
npm install -g <package_name>
```

3.6.1 How to Uninstall Global Packages

- For you to uninstall a package all you need to do is to type:

```
npm uninstall -g <package>
```

- If you want to uninstall a package called jshint, you would type:

```
npm uninstall -g jshint
```

3.7 UPDATING PACKAGES

Updating Local Packages:

- You should periodically update the packages that your application depends on. Then if there are code changes made by the original developers, your code will also be improved.
- To do this:
  1. Navigate to the root directory of your project and ensure it contains a Package.json file:

```
cd /path/to/project
```

  2. In your project root directory, run the update command:

```
npm update
```

  3. To test the update, run the outdated command. There should not be any output.

```
npm outdated
```

Updating Global Packages

- This requires version 2.6.1 or greater.
- If you want to update packages, you should type this command on your terminal:

```
npm update -g <package>
```

- For instance, if you want to update a package called grunt, you would type:

```
npm update -g grunt
```

- If you want to find out the packages that needs to be updated, type:

```
npm outdated -g --depth=0
```

- Finally, if you want to update all global packages, you should type:

```
npm update -g
```

- For any npm version that is below 2.6.1, you should run this script:

```
#!/bin/sh
set -e
set -x
for package in $(npm -g outdated --parseable --depth=0 | cut -d: -f3)
do
npm -g install "$package"
done
```

To update all outdated global packages.

- However, it is recommended that you upgrade to the latest version of npm. You can do this by typing:

```
npm install npm@latest -g
```

3.8 MANAGING THIRD PARTY PACKAGES WITH NPM

- As we have seen, the "Node package manager" has the ability to manage modules, which are required by Node JS applications.
- Let's look at some of the functions available in the node package manager for managing modules.

1. **Installing packages in global mode:** Modules can be installed at the global level, which just basically means that these modules would be available for all Node JS projects on a local machine. The below example shows, how to install the "express module" with the global option.

```
npm install express -global
```

The global option in the above statement is what allows the modules to be installed at a global level.

2. **Listing all of the global packages installed on a local machine:** This can be done by executing the below command in the command prompt:

```
npm list --global
```

Below is the output which will be shown, if you have previously installed the "express module" on your system. Here you can see the different modules installed on the local machine.

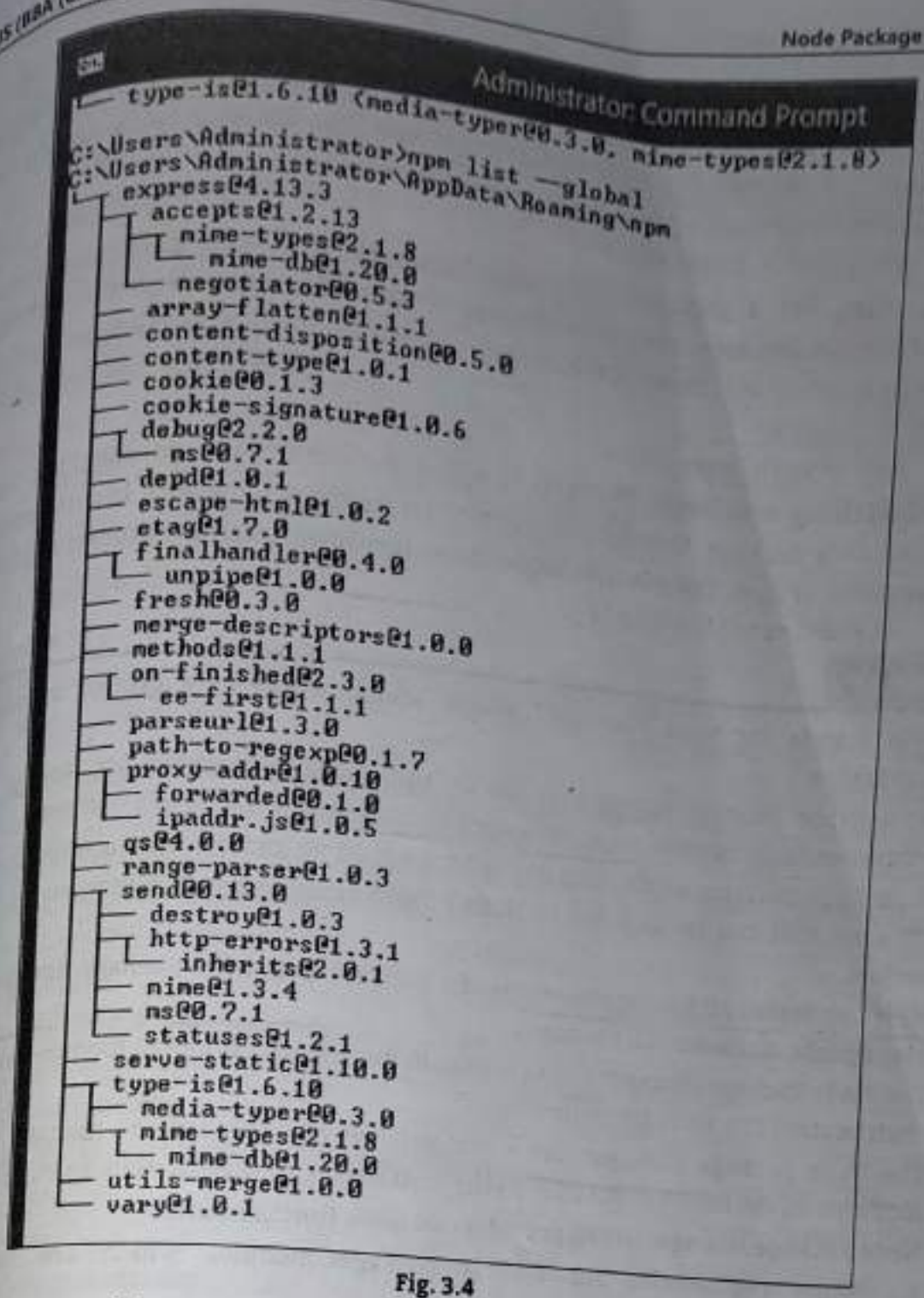


Fig. 3.4

Installing a specific version of a package: Sometimes there may be a requirement to install just the specific version of a package. Once you know what are the package and the relevant version that needs to be installed, you can use the npm install command to install that specific version. The example below shows how to install the module called underscore with a specific version of 1.7.0.

```
npm install underscore@1.7.0
```

4. **Updating a package version:** Sometimes you may have an older version of a package in a system, and you may want to update to the latest one available in the market. To do this one can use the npm update command. The example below shows how to update the underscore package to the latest version.

```
npm update underscore
```

5. **Searching for a particular package:** To search whether a particular version is available on the local system or not, you can use the search command of npm. The example below will check if the express module is installed on the local machine or not.

```
npm search express
```

6. **Uninstalling a package**: The same in which you can install a package, you can also uninstall a package. The uninstallation of a package is done with the uninstallation command of npm. The example below shows how to uninstall the express module.

```
npm uninstall express
```

Summary

- NPM stands for Node Package Manager and it works as a project manager for JavaScript.
- NPM (Node Package Manager) is the default package manager for Node.js and is written entirely in JavaScript. Developed by Isaac Z.
- A package contains all the files needed for a module and modules are the JavaScript libraries that can be included in Node project according to the requirement of the project.
- NPM can install all the dependencies of a project through the package.json file. It can also update and uninstall packages.
- The Node Package Manager (npm) is used to download and install modules which can then be used in a Node JS application.
- The Node package manager has a complete set of commands to manage the npm modules on the local system such as the installation, un-installation, searching, etc
- Node Package Manager (NPM) provides two main functionalities:
  1. Online repositories for node.js packages/modules which are searchable on search.nodejs.org.
  2. Command line utility to install Node.js packages, do version management and dependency management of Node.js packages.
- Version of npm installed on system can be checked using `npm -v`.
- To install packages and modules in the project use the `npm install package_name` syntax.

- To install a package globally, add an extra -g tag in syntax like `npm install package_name -g`.
- To install a package and simultaneously save it in package.json, add –save flag. The –save flag is default in npm install command so it is equal to `npm install package_name` command.
- To uninstall packages using npm, use the syntax: `npm uninstall`
- To uninstall global packages, follow the syntax: `npm uninstall package_name -g`

Check Your Understanding

1. Node JS is written in which language_____.
   (a) JavaScript (b) C
   (c) C++ (d) All of the above
2. Which function is used to include modules in Node JS?
   (a) include( ); (b) require( );
   (c) attach( ); (d) None of the above
3. Which statement executes the code of sample.js file?
   (a) nodejs sample.js (b) node sample.js
   (c) sample.js (d) None of the above
4. The $ npm ls -g statement is used to list down all the locally installed module?
   (a) True (b) False
5. How can we check the current version of NPM?
   (a) npm –version (b) npm --ver
   (c) npm help (d) None of the above
6. To install Node JS express module_____is used.
   (a) $ npm install express (b) $ node install express
   (c) $ install express (d) None of above
7. What is the default scope in Node.js application?
   (a) Global (b) Local
   (c) Public (d) Private
8. For what npm stands?
   (a) Node Project Manager (b) Node Package Manager
   (c) New Project Manager (d) New Package Manager
9. Command to list all modules that are install globally?
   (a) $ npm ls -g (b) $ npm ls
   (c) $ node ls -g (d) $ node ls

10. Which of the following is true about package.json?
    (a) package.json is present in the root directory of any Node application/module
    (b) package.json is used to define the properties of a package.
    (c) package.json can be used to update dependencies of a Node application.
    (d) All of the above.

ANSWER KEY

1. (d)	2. (b)	3. (b)	4. (b)	5. (a)
6. (a)	7. (b)	8. (b)	9.(a)	10. (d)

Practice Questions

Q.I: Answer the following questions in short.

1. Write syntax to install local package? – npm install <package name>
2. Write syntax to install global package? – npm install <package name>
3. How can you check the installed version of Node JS?
4. For what require( ) is used in Node JS?
5. List out NPM components.
6. What is use of Registry?

Q.II: Answer the following questions.

1. What is npm? Explain with its advantages.
2. What is the Package.json file? What is it used for?
3. Define the functionalities of NPM?
4. Explain some popular modules in Node JS?
5. How to add dependency into package.json?

Q.III: Define the terms.

1. NPM
2. Package
3. Json Package
4. Local Package
5. Global package

❖❖❖

4...

Objectives...

- To learn about Creating Web S
- To study Handling HTTP Requ
- To learn about Sending Reque

4.1 INTRODUCTION

- **Web Server:** The word web serv working simultaneously.

1. **Hardware Part:** A hardwa software and a website's images, CSS stylesheets, Internet and supports req devices connected to the w
2. **Software Part:** Software files on web servers. On server (HTTP). An HTT locator (web addresses) a HTML web page's). An H the websites it stores, a end user's device.

To publish a website, you need

1. **Static Web Server:**

- A static web server consist it is known as static becau runtime data processing Static Web Server.
- **Static Website:** Static we developed in small time

4...

Web Server

Objectives...

- To learn about Creating Web Server.
- To study Handling HTTP Request.
- To learn about Sending Request.

4.1 INTRODUCTION

- **Web Server:** The word web server comprises of hardware or software, or both of them working simultaneously.

1. **Hardware Part:** A hardware web server is a computer that stores web server software and a website's component files. (For example, HTML documents, images, CSS stylesheets, and JavaScript files). A web server connects to the Internet and supports request response in terms of information exchange to the devices connected to the web server.
2. **Software Part:** Software part of web server, manages web users access hosted files on web servers. On current context this is a Hyper Text Transfer Protocol server (HTTP). An HTTP server is software that executes uniform resource locator (web addresses) and HTTP instructions (the protocol browser uses to view HTML web page's). An HTTP server can be accessed through the domain names of the websites it stores, and it delivers the content of these hosted websites to the end user's device.

To publish a website, you need either a Static or Dynamic web server.

1. **Static Web Server:**

- A static web server consists of a computer (hardware) with an HTTP server (software). It is known as static because the server sends its hosted files as it is to browser. Any runtime data processing or execution of instructions cannot be implemented on Static Web Server.
- **Static Website:** Static website is considered as one of the easy website which can be developed in small time. It consists of limited number of pages. Sometime it is of

single page which is like landing page. That page consists of all the information related to your business services.

- **Advantages of Static Websites:**
  1. Convenient to Develop.
  2. Low cost.
  3. Convenient and cheap to host.
- **Disadvantages of Static Websites:**
  1. It requires skilled web developer who can update the site versions.
  2. It requires a strong Search Engine Optimization professional in order to resolve all the technical issues from Google point of view.
  3. Static websites are not end user friendly.

2. **Dynamic Web Server:**

- A dynamic web server comprises of a static web server and an application server and a database. It is known as dynamic because the application server updates the hosted files before sending content to your browser via the HTTP server.
- **Dynamic Website:** Dynamic websites are known as one of the best website from customer point of view. Dynamic websites are expensive to develop and deploy. Dynamic websites are search engine friendly and easy to update. These website has inbuilt content management system where content can be updated as per need. Now a day's, most of the websites contains dynamic server. For example banking, insurance and airline reservation web applications.
- **Advantages of Dynamic Website:**
  1. Dynamic content development and updating on various static pages of websites.
  2. Very easy to manage and update.
  3. You can update user friendly content related to your services or blogs and rank them in search engines.
- The main **disadvantage** of having dynamic website is they are costly and even their hosting is costly.
- **Static Vs. Dynamic Website:** Following table shows difference between Static Website and Dynamic Website

Table 4.1: Static Vs Dynamic Website

Sr. No.	Static Website	Dynamic Website
1.	When page is loaded the prebuilt content is same every time.	Content is generated quickly and changes regularly.

Contd...

2.	It uses the HTML code for developing a website.	It uses the server side languages such as PHP, SERVLET, JSP, and ASP.NET etc. for developing a website.
3.	It sends exactly the same response for every request.	It may generate different HTML for each of the request.
4.	The content is only changed when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code which allows the server to generate the unique content when the page is loaded.
5.	Flexibility is the main advantage of static website.	Content Management System (CMS) is the main advantage of dynamic website.

4.1.1 HTTP Request Response

- On a web server, the HTTP server is responsible for processing and answering incoming requests.
- When browser or a client needs a webpage that is hosted on a web server, the browser requests the file through HTTP. When the request reaches the correct web server (Hardware or IP address), the (software) HTTP server accepts the request, process the requested document, and sends it back to the browser, also through HTTP. (If the server doesn't find the requested document, it returns a 404 response instead.)

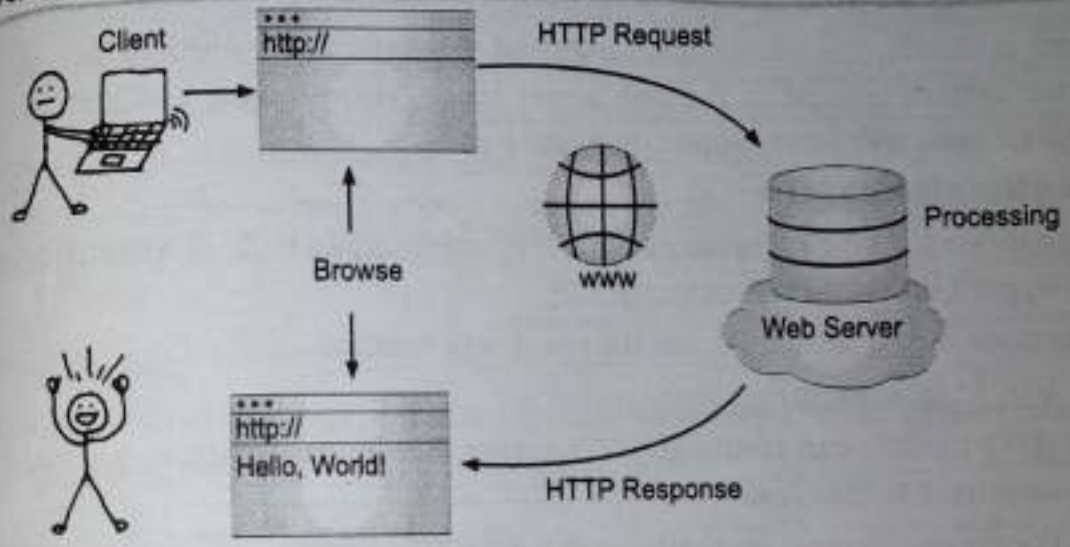


Fig. 4.1: HTTP Request and HTTP Response

4.1.2 Features of HTTP

- **HTTP is Stateless:** HTTP is called as a stateless protocol because there is no link between two requests being successively carried out on the same connection. This immediately has the prospect of being problematic for users attempting to interact with certain pages coherently, for example, using e-commerce shopping baskets. But

while the core of HTTP itself is stateless, HTTP cookies allow the use of stateful sessions. Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.

- **HTTP is Connection less:** The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server know about each other during current request and response only. Further requests are made on new connection like client and server are new to each other.
- **HTTP is Simple:** HTTP is generally designed to be simple and human readable even with the added complexity introduced in HTTP/2 by encapsulating HTTP messages into frames. HTTP messages can be read and understood by humans, providing easier testing for developers, and reduced complexity for newcomers.
- **HTTP is extensible:** HTTP can be integrated with new functionality by providing a simple agreement between a client and a server.

4.2 CREATING WEB SERVER

- The Node JS Framework is mostly used to create server based applications. The Framework can easily used to create web service which can server content to user.
- There is variety of models such as **http** and **request** module which helps in processing server related request in the web server space. We will have a look at how we can create a basic web server application using node JS.

Creating Node JS Web Server:

1. Node JS has a built-in module called HTTP, which allows Node JS to transfer data over the Hyper Text Transfer Protocol (HTTP).
2. To include the HTTP module, use the `require()` method:

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

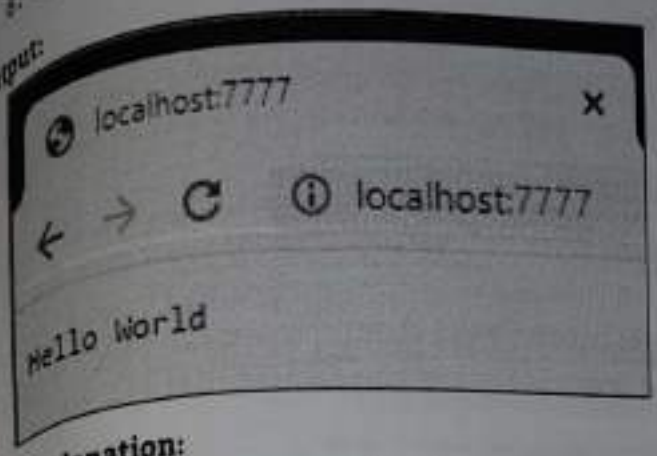
3. The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
4. Use the `createServer()` method to create an HTTP server.

- Let's look at an example how to create and run node JS application.
- Our application is going to create a simple server module which will listen on port number *7777* if a request is made through the browser on this port number then server application will send a "Hello World" response to the client.

```
1. var http = require('http')
2. var server = http.createServer((function (request, response) {
```

```
response.writeHead(200, { "Content-Type": "text/plain" });
response.end("Hello World\n");
});
server.listen(7777);
```

Output:



Code explanation:

1. We import the http module using require() function. The http module is a core module of Node JS, so no need to install it using NPM.
2. The next step is to call createServer() method of http and specify callback function with request and response parameter.
3. When a request is received, we are saying to send a response with the header type of 200 this number is normal response which is sent in an HTTP header when a successful response is sent to the client.
4. In the response itself we are sending the string Hello World.
5. We are then using the server.listen() function to make our server application listen to the client request on port number 7777. You can specify any available port over here.

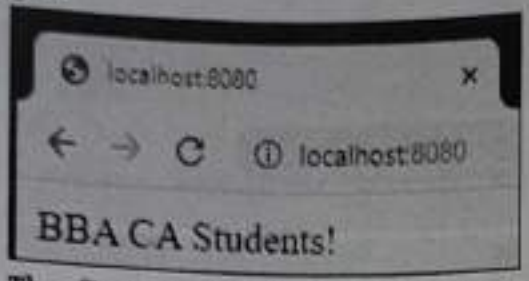
Add an HTTP Header:

- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type.

Program 4.1: Program to add an http header.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('BBA CA Students!');
  res.end();
}).listen(8080);
```

Output:



- The first argument of the `res.writeHead()` method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

4.3 HANDLING HTTP REQUESTS

- The `http.createServer()` method includes request and response parameters which is supplied by Node JS.
- The request object can be used to get information about the current HTTP request e.g. URL, request header, and data. The response object can be used to send a response for a current HTTP request.

Program 4.2: Program demonstrates handling HTTP request and response in Node JS.

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

var server = http.createServer(function (req, res) {
    if (req.url == '/') {
       res.writeHead(200, { 'Content-Type': 'text/html' });
       res.write('<html><body><p>This is home Page.</p></body></html>');
       res.end();
    }

   else if (req.url == "/student") {
       res.writeHead(200, { 'Content-Type': 'text/html' });
       res.write('<html><body><p>This is student Page.</p></body></html>');
       res.end();
    }

   else if (req.url == "/admin") {
       res.writeHead(200, { 'Content-Type': 'text/html' });
       res.write('<html><body><p>This is admin Page.</p></body></html>');
       res.end();
   }

   else
```

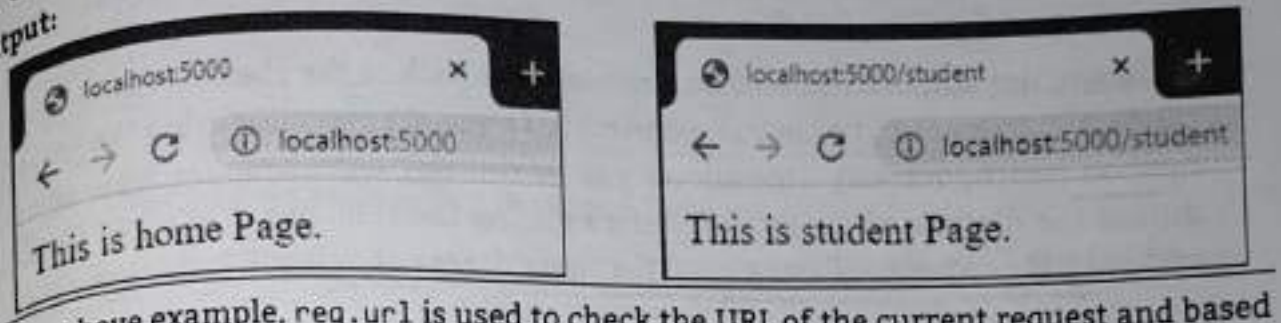
```
    res.end('Invalid Request!');

});

server.listen(5000);

console.log('Node JS  web server at port 5000 is running..')
```

Output:



- The above example, `req.url` is used to check the URL of the current request and based on that it sends the response. To send a response, first it sets the response header using `writeHead()` method and then writes a string as a response body using `write()` method. Finally, Node JS web server sends the response using `end()` method.
- Now, run the above web server as shown below.

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Node JS  web server at port 5000 is running.
```

- **1st Output Screen:** For Windows users, point your browser to `http://localhost:5000` and see the 1st output screen.
- **2nd Output Screen:** The same way, point your browser to `http://localhost:5000 student` and see 2nd output screen.

4.3.1 How Node Presents Incoming HTTP Requests to Developers

- Node provides HTTP server and client interfaces through the http module:

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

- To create an HTTP server, call the `http.createServer()` function. It accepts a single argument, a callback function that will be called on each HTTP request received by the server. This request callback receives as arguments, the request and response objects, which are commonly shortened to `req` and `res`:

```
var http = require('http');
var server = http.createServer(function(req, res){
// handle request
});
```

- For every HTTP request received by the server, the request callback function will be invoked with new req and res objects. Prior to the callback being triggered, Node will parse the request up through the HTTP headers and provide them as part of the req object. But Node doesn't start parsing the body of the request until the callback has been fired. This is different from some server side frameworks, like PHP, where both the headers and the body of the request are parsed before your application logic runs. Node provides this lower level interface so you can handle the body data as it is being parsed, if desired.
- Node will not automatically write any response back to the client. After the request callback is triggered, it's your responsibility to end the response using the `res.end()` method (see figure 4.2). This allows you to run any asynchronous logic you want during the lifetime of the request before ending the response. If you fail to end the response, the request will hang until the client times out or it will just remain open.

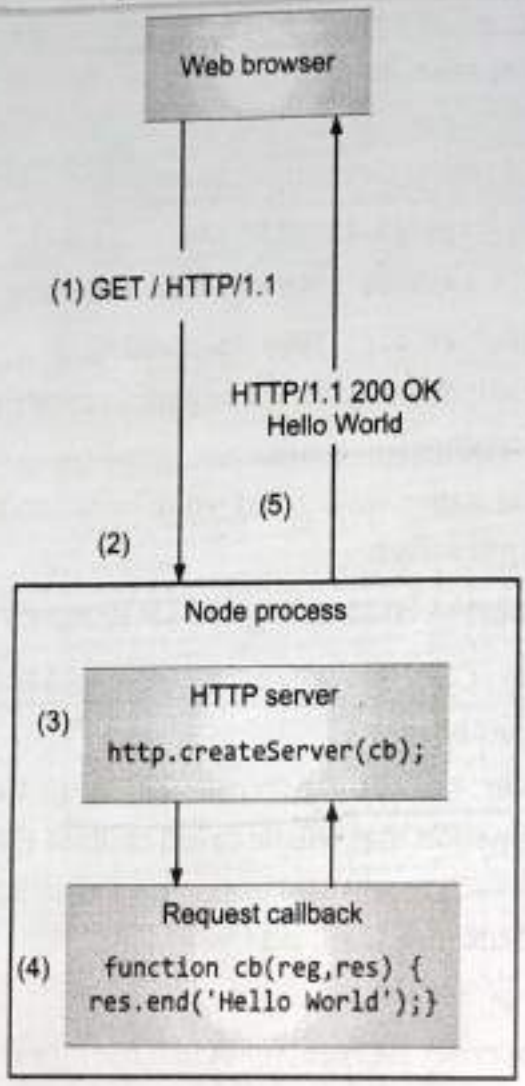


Fig. 4.2: The lifecycle of an HTTP request going through a Node HTTP server

(1) An HTTP client like a web browser initiates an HTTP request.
(2) Node accepts the connection, and incoming request data is given to the HTTP server.
(3) The HTTP server parses up to the end of the HTTP headers and then hands control over to the request callback.
(4) The request callback performs application logic, in this case responding immediately with the text "Hello World".
(5) The request is sent back through the HTTP server, which formats a proper HTTP response for the client.

- Node servers are long-running processes that serve many requests throughout their lifetimes.

Reading Request Headers and Setting Response Headers:

- The Hello World example in the previous section demonstrates the bare minimum required for a proper HTTP response. It uses the default status code of 200 (indicating success) and the default response headers. Usually, though, you'll want to include any number of other HTTP headers with the response. For example, you'll have to send a Content-Type header with a value of text/html when you're sending HTML content so that the browser knows to render the result as HTML.
- Node offers several methods to progressively alter the header fields of an HTTP response: `res.setHeader(field, value)`, `res.getHeader(field)`, `res.removeHeader(field)` methods. Here's an example of using `res.setHeader()`:

```
var body = 'Hello World'; res.setHeader('Content-Length', body.length);
res.setHeader('Content-Type', 'text/plain'); res.end(body);
```

- You can add and remove headers in any order, but only up to the first `res.write()` or `res.end()` call. After the first part of the response body is written, Node will flush the HTTP headers that have been set.

4.4 SENDING REQUESTS

- The `req` parameter of the `http.createServer()` method has all the information about the incoming request for eg. request payload, headers, URL etc.
- The `req` parameter has the following properties:

1. **request.headers:** Information about the request headers such as Connection, Host, Authorization, etc.
2. **request.method:** Information about the incoming request methods such as GET, POST, PUT, DELETE, OPTIONS, HEAD, etc.
3. **request.url:** Information about the incoming request URL, such as /accounts, /users, /messages etc.

Request Method:

a. GET

- Most requests are GET requests Node JS provides this method. The only difference between this method and `http.request()` is that it sets the method to GET and calls `req.end()` automatically.
- **Example:**

```
const https = require('https')
const options = {
  hostname: 'anyURL.com',
  port: 450,
  path: '/myfolder',
  method: 'GET'
}

const req = https.request(options, res => {
console.log('statusCode: ${res.statusCode}')

res.on('data', d => {
process.stdout.write(d)
  })
})

req.on('error', error => {
console.error(error)
})
req.end()
```

b. POST

- `http.request()` returns instance `http.ClientRequest` class.
- The ClientRequest instance is a writable stream. If one needs to upload a file with a POST request, then write to the ClientRequest object.
- **Example:**

```
const https = require('https')

const data = JSON.stringify({
  todo: 'Buy the milk'
})
```

```
const options = {
  hostname: 'anyURL.com',
  port: 450,
  path: '/myfolder',
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': data.length
  }
}

const req = https.request(options, res => {
console.log('statusCode: ${res.statusCode}')

res.on('data', d => {
process.stdout.write(d)
  })
})

req.on('error', error => {
console.error(error)
})

req.write(data)
req.end()
```

c. **PUT**

PUT requests use the same POST request format and just change the options method value.

- **Example:**

```
const https = require('https')

const data = JSON.stringify({
  todo: 'Buy the milk'
})
```

```
const options = {
  hostname: 'anyURL.com',
  port: 450,
  path: '/myfolder',
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': data.length
  }
}

const req = https.request(options, res => {
console.log('statusCode: ${res.statusCode}')

res.on('data', d => {
process.stdout.write(d)
  })
})

req.on('error', error => {
console.error(error)
})

req.write(data)
req.end()
```

d. DELETE

DELETE requests use the same POST request format and just change the options method value.

- **Example:**

```
const https = require('https')

const data = JSON.stringify({
  todo: 'Buy the milk'
})
```

```
const options = {
  hostname: 'anyURL.com',
  port: 450,
  path: '/myfolder',
  method: 'DELETE',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': data.length
  }
}

const req = https.request(options, res => {
console.log('statusCode: ${res.statusCode}')

res.on('data', d => {
process.stdout.write(d)
  })
})

req.on('error', error => {
console.error(error)
})

req.write(data)
req.end()
```

e. **HEAD**

- The response to an HTTP HEAD request is always just the response header. The response body is always 0 length.
- **Example:**

```
'use strict';

const http = require('http');

http.request('http://example.com', { method: 'HEAD' }, (res) => {
  console.log(res.statusCode);
}).on('error', (err) => {
console.error(err);
}).end();
```

Summary

- **Web Server:** The word web server comprises of hardware or software, or both of them working simultaneously.
- **Hardware Part:** A hardware web server is a computer that stores web server software and a website's component files.
- **Software Part:** Software part of web server, manages web users access hosted files on web servers. On current context this is a Hyper Text Transfer Protocol server (HTTP).
- A static web server consists of a computer (hardware) with an HTTP server (software). It is known as static because the server sends its hosted files as it is to browser.
- Static website is considered as one of the easy website which can be developed in small time. It consists of limited number of pages.
- A dynamic web server comprises of a static web server and an application server and a database. It is known as dynamic because the application server updates the hosted files before sending content to your browser via the HTTP server.
- Dynamic web sites are known as one of the best website from customer point of view. Dynamic websites are expensive to develop and deploy. Dynamic websites are search engine friendly and easy to update.
- There is variety of models such as http and request module which helps in processing server related request in the web server space.

Check Your Understanding

1. The word web server comprises which of the following?
   (a) hardware (b) software
   (c) Both (d) None of the above
2. Which is not advantage of static websites?
   (a) Convenient to develop (b) Low cost.
   (c) Convenient and cheap to host. (d) High cost
3. HTTP is _____.
   (a) Stateless (b) Stateful
   (c) Both the above (d) None of the above
4. Which method includes request and response parameters which are supplied by Node JS?
   (a) http.createServer()
   (b) http.MakeServer()
   (c) http.GetServer()
   (d) None of the above

5. Difference between GET method and http.request() is_____.
   (a) It sets the method to GET and calls req.end() automatically.
   (b) It sets the method to GET and calls req.terminate() automatically.
   (c) It sets the method to GET and calls req.finish() automatically
   (d) None of the above
6. HTTP is_____.
   (a) Connection less
   (b) Connection oriented
   (c) Both a & b
   (d) None of the above
7. PUT, POST and which requests use the same request format and just change the options method value.
   (a) DELETE
   (b) GET
   (c) HEAD
   (d) None of the above
8. The main disadvantages of having dynamic website is _____.
   (a) They are costly and even their hosting is costly.
   (b) Less costly
   (c) Hosting is less costly
   (d) None of the above

ANSWER KEY

1. (c)	2. (d)	3. (a)	4. (a)	5. (a)
6. (a)	7. (a)	8. (a)		

Practice Questions

Q.I Answer the following questions in short.

1. List out sending request methods. – Get, put, delete, head, post,
2. What is difference between GET and http.request() method?
3. Explain working of writeHead().
4. Explain advantages and disadvantages of static website.
5. Explain advantages and disadvantages of dynamic website.

Q.II Answer the following questions.

1. Explain web server with real example?
2. Write difference between static website and dynamic website.

3. How to read query string, write sample code and explain it?
4. How to split query string, take example of splitting first name and last name from given user name?
5. Explain the function of requestListener() function with suitable example?
6. How to read text from web pages?
7. Write Steps to create web Server?
8. What are functions of HTTP?

Q.III Define the terms.

1. Web site
2. Web Server
3. Static website
4. Dynamic website

❖❖❖

5...

File System

Objectives...

- To learn the operations on file.
- To learn about I/O operations.
- To study about Handling Exceptions and Errors.

5.1 INTRODUCTION

About Node JS file System:

- In this section, we discuss handling files from the file system which is important for loading and parsing files in your application.
- The file system is big part of any application need to handle files for loading, manipulating or serving data.
- We can handle file operations such as create, read, delete and many more using Node JS. The built-in model provided by Node JS is `fs` (File System).
- Node JS provides functionality of file I/O by providing wrappers around the standard functions.
- Depending upon user requirements. File system operations can have synchronous and asynchronous approach.
- To use this file system (`fs`) module, use the require() method:

```
var fs = require('fs');
```

5.1.1 Synchronous and Asynchronous Approach

Synchronous Approach:

- We also called Synchronous approach as **"blocking functions"** as after completion of one operation it starts the next operation i.e. to execute next command previous all command must have executed.
- For Example, Imagine the situation of a restaurant. The waiter will take order from table1 give it to the kitchen and will wait until the chef prepares the food. This is called **blocking or synchronous nature.**

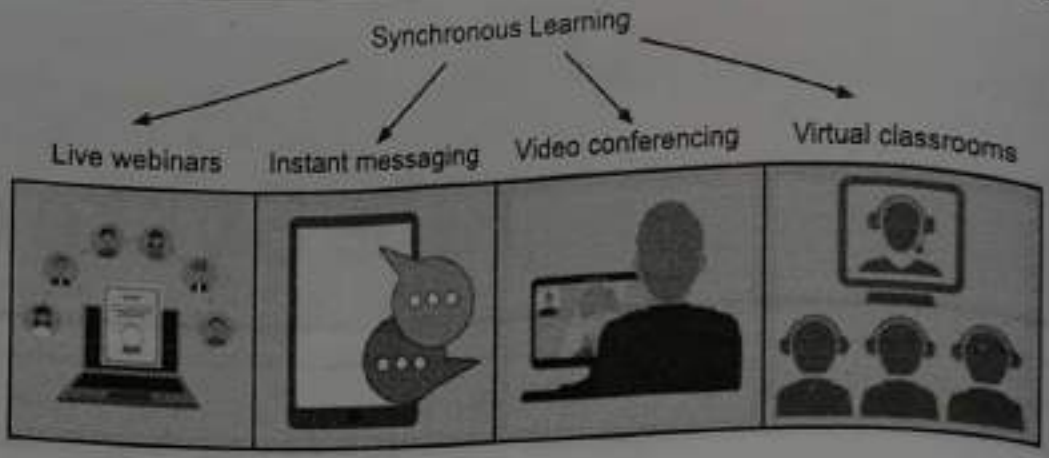


Fig. 5.1: Synchronous Learning

Asynchronous Approach:

- We also called Asynchronous approach as **"non-blocking functions"**. As it never waits other operations to complete. It may happen that several commands are executing in background and several are executing in foreground.
- For Example, in contrast the waiter takes order from table1 and gives the order in the kitchen, now the waiter can serve another table let's say table2 and get their order. This is **asynchronous system**, as the chef is preparing the food, meanwhile a single waiter is serving different tables (taking orders and serving food). The waiter doesn't have to wait for the chef to cook the meal before he has to go to another table. This is what we call a **non-blocking** or **asynchronous** architecture.

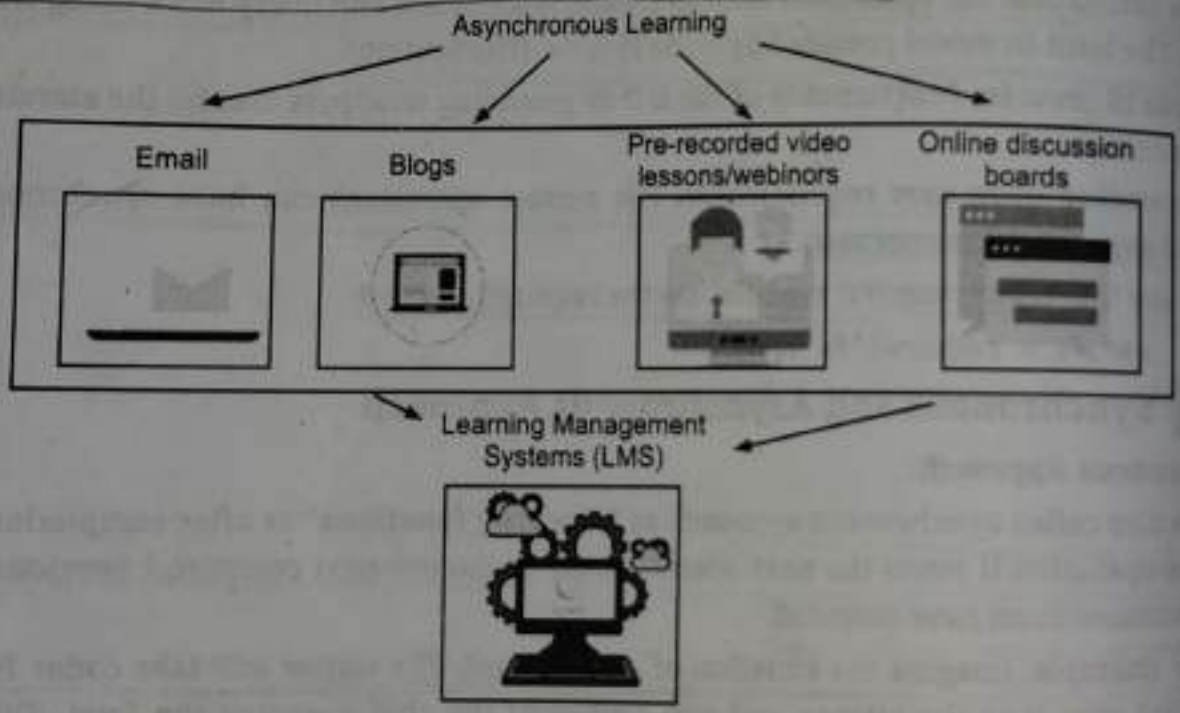


Fig. 5.2: Asynchronous learning

Difference between Synchronous and Asynchronous Approach:

Table 5.1: Difference between Synchronous and Asynchronous

Sr. No.	Synchronous	Asynchronous
1.	Students learn at the same time.	Students learn at different time.
2.	Communication happens in real time.	Communication is not live.
3.	Possibly more engaging and effective.	Possibly more convenient and flexible.
4.	Allows for instant feedback and clarification.	Allow students to work at their own speed.
5.	Examples: Video conferencing, live chat, live streamed videos.	Examples: Email, Screencasts, Flipgrid, Videos, Blog posts/comments.

Implementation of Synchronous and Asynchronous Modes in Various Applications:

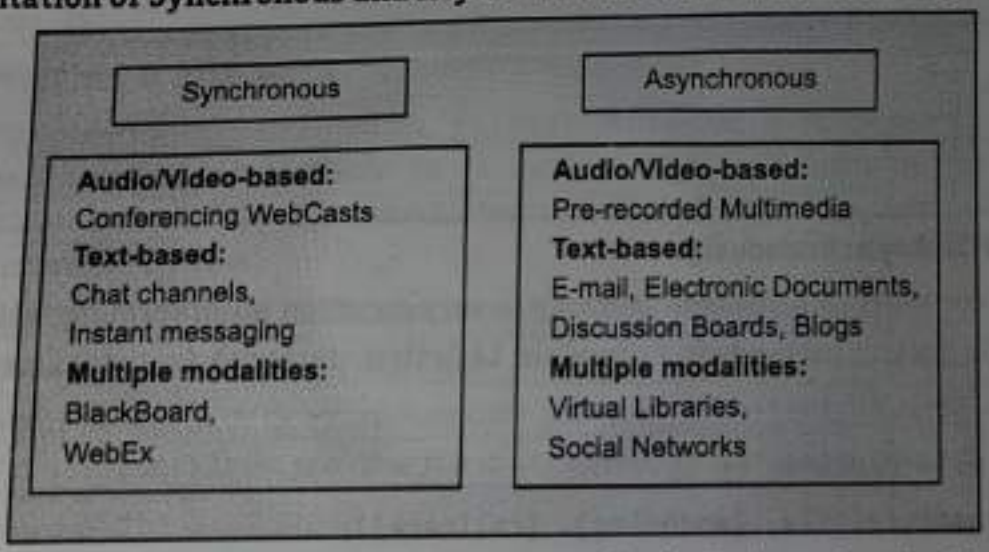


Fig. 5.3: Synchronous and Asynchronous Modes in Various Applications

5.2 OPERATIONS ON FILE

5.2.1 Read a File (fs.readFile)

1. **Read a File Synchronously:**

- The `fs.readFileSync()` method is an inbuilt application programming interface of `fs` module which is used to read the file and return its content.
- In `fs.readFileSync()` method, we can read files in a synchronous way, i.e. we are telling Node JS to block other parallel process and do the current file reading process.

- **Syntax:**

```
fs.readFileSync(path, options)
```

Where,

path: It takes the relative path of the text file.

options: It is an optional parameter which contains the encoding and flag.

Program 5.1: Node JS program to demonstrate the fs.readFileSync() method.

```
const fs = require('fs');
// Call readFileSync() method to read 'programming.txt' file
const data = fs.readFileSync('./programming.txt',{encoding:'utf8', flag:'r'
});
// Display the file data
console.log(data);
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
This is a file containing a collection of programming languages.
1. C
2. C++
3. JAVA
4. Angular JS
5. Node JS
```

2. Read a File Asynchronously:

- To read contents of a file into memory is very common programming task. There are many file system methods in `fs` module. Let's first start with `fs.readFile()`.
- **Syntax:**

```
var fs = require('fs');
fs.readFile(file, [encoding], [callback]);
```

Where,

file: File path of the file or file name.

encoding is an optional parameter that specifies the type of encoding to read the file. Types of encodings are 'ascii', 'utf8', and 'base64'. The default encoding is null.

callback is a function to call when the file has been read and the contents are ready. It is passed two arguments, error and data.

- For example, suppose we have predefined file `calc.js` having some content. Now we will open this file and read file by writing code into `app.js` file.

Program 5.2: Program for fs.readFile().

```
fs = require('fs');
fs.readFile('programming.txt', 'utf8', function (err,data) {
   if (err) {
      return console.log(err);
   }
   console.log(data);
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
This is a file containing a collection of programming languages.
1. C
2. C++
3. JAVA
4. Angular JS
5. Node JS
```

5.2.2 Writing a File Data

- File I/O operation implemented by 'fs' module of Node JS. There are synchronous as well as asynchronous methods in fs module. The asynchronous function has a callback function. The last parameter which indicates the completion of the asynchronous function.
- There are 2 ways to write in file synchronously and asynchronously.

1. Synchronously Writing Data to a File:

- If we want to write data synchronously we use `fs.writeFileSync()` method. The `fs.writeFileSync()` creates a new file if the specified file does not exist.
- **Syntax:**

```
fs.writeFileSync(file, data, option)
```

Where,

file: It is a string, Buffer, URL or file description integer that indicates the path of the file where it has to be written. Using a file descriptor will make the work similar to fs.write() method.

data: Buffer, TypedArray or DataView that will be written to the file. The data is of string type.

options: It is a string or object that can be used to specify optional parameters that will affect the output. It has three optional parameters:

- **encoding:** It is a string which specifies the encoding of the file. The default value is 'utf8'.
- **mode:** It is an integer which specifies the file mode. The default value is 0o666.
- **flag:** It is a string which specifies the flag used while writing to the file. The default value is 'w'.

Program 5.3: Node JS program to demonstrate the fs.writeFileSync() method.

```
const fs = require('fs');
let data = "This is a file containing a collection"
        + " of programming languages.\n"
        + "1. JAVA\n2. C++\n3. R";
fs.writeFileSync("programming.txt", data);
console.log("File written successfully\n");
console.log("The written has the following contents:");
console.log(fs.readFileSync("programming.txt", "utf8"));
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
File written successfully
The written has the following contents:
This is a file containing a collection of programming languages.
1. JAVA
2. C++
3. R
```

2. Asynchronously Writing Data to a File:

- If we want to write data asynchronously we use `fs.writeFile()` method.
- **Syntax:**

```
fs.writeFile( file, data, options, callback )
```

Where,

file: It is a string, Buffer, URL or file description integer that denotes the path of the file where it has to be written.

data: It is a string, Buffer, TypedArray or DataView that will be written to the file.

options: It is a string or object that can be used to specify optional parameters that will affect the output. It has three optional parameters:

- **encoding:** It is a string value that specifies the encoding of the file. The default value is 'utf8'.

- **mode:** It is an integer value that specifies the file mode. The default value is 0o666.
- **flag:** It is a string value that specifies the flag used while writing to the file. The default value is 'w'.

  callback: It is the function that would be called when the method is executed.
- **err:** It is an error that would be thrown if the operation fails.

Program 5.4: Node JS program to demonstrate the fs.writeFile() method.

```
const fs = require('fs');
let data = "This is a file containing a collection of books.";

fs.writeFile("books.txt", data, (err) => {
  if (err)
    console.log(err);
  else {
    console.log("File has been written successfully\n");
    console.log("With following contents:");
    console.log(fs.readFileSync("books.txt", "utf8"));
  }
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
File has been written successfully
With following contents:
This is a file containing a collection of books.
```

5.2.3 Opening a File

- To open a file, create file, to write to a file or to read a file `fs.open()` method is used. `fs.open()` method performs several operations on a file. Initially we have to upload the `fs` class, which is module to access physical file system.
- **Syntax:**

```
fs.open( filename, flags, mode, callback )
```

Where,

filename: It holds the name of the file to read or the entire path if stored at other location.

flag: The operation in which file has to be opened.

`mode`: Sets the mode of file i.e. r-read, w-write, r+ -readwrite. It sets to default as readwrite.

`callback`: It is a callback function that is called after reading a file. It takes two parameters:

- `err`: If any error occurs.
- `data`: Contents of the file. It is called after open operation is executed.

- All types of flags are described below:

Table 5.2: Flags

FLAG	DESCRIPTION
r	This flag will open file to read and throws exception if file doesn't exist.
r+	This flag will open file to read and write. Throws exception if file doesn't exist.
rs+	This flag will open file in synchronous mode to read and write.
w	It is use to open file for writing. File is created if it doesn't exist.
wx	It is same as 'w' but fails if path exists.
w+	It is use to open file to read and write. File is created if it doesn't exist.
wx+	It is same as 'w+' but fails if path exists.
a	It is use to open file to append. File is created if it doesn't exist.
ax	It is same as 'a' but fails if path exists.
a+	It is use to open file for reading and appending. File is created if it doesn't exist.
ax+	It is same as 'a+' but fails if path exists.

Program 5.5: Node JS program to demonstrate the fs.open() Method in write mode.

```
var fs = require('fs');
// To open file in Write mode, create file if doesn't exists.
fs.open('demo.txt', 'w', function (err, f) {
    if (err) {
        return console.error(err);
    }
      console.log(f);
      console.log("File opened!!");
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
3
File opened!!
```

Program 5.6: Node JS program to demonstrate the fs.open( ) Method for read mode.

```
var fs = require('fs');
//open file in read mode, exception occurs if the file does not exist
fs.open('demo.txt', 'r', function (err, f) {
   if (err) {
      return console.error(err);
   }
   console.log(f);
   console.log("File opened!!");
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
3
File opened!!
```

5.2.4 Deleting a File

- Whenever we want to delete a file we use `fs.unlink()` method. This method does not work on directories. To delete directories we have to use `fs.rmdir()`.
- **Syntax:**

```
fs.unlink(path,callback)
```

Where,

path: It is a string, Buffer or URL which represents the file or symbolic link which has to be removed.

callback: It is a function that would be called when the method is executed.

- **err:** It is an error that would be thrown if the method fails.

Program 5.7: Program to remove a file from the file system.

```
const fs = require('fs');
// Get the files in current directory before deletion
getFilesInDirectory();

// Delete books.txt
fs.unlink("books.txt", (err => {
  if (err) console.log(err);
```

```
    else {
      console.log("\nDeleted file: books.txt");
      // Get the files in current directory after deletion
      getFilesInDirectory();
    }
  }));
  //Function to get current filenames in directory with specific extension
  function getFilesInDirectory() {
    console.log("\nFiles present in directory:");
    let files = fs.readdirSync(__dirname);
    files.forEach(file => {
      console.log(file);
    });
  }
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Files present in directory:
app.js
books.txt
demo.txt
node_modules
package-lock.json
package.json
programming.txt

Deleted file: books.txt

Files present in directory:
app.js
demo.txt
node_modules
package-lock.json
package.json
programming.txt
```

5.2.5 Truncate a File

- The `fs.ftruncate()` method is used to change the size of the file i.e. either increase or decrease the file size.
- It changes the length of the file at the path by `len` bytes. If `len` is shorter than the file's current length, the file is truncated to that length. If it is greater than the file length, it is padded by appending null bytes (x00) until `len` is reached. It is similar to the truncate() method, except it accepts a file descriptor of the file to truncate.
- **Syntax:**

```
fs.ftruncate(fd, len, callback)
```

Where,

fd: This is the file descriptor returned by fs.open().

len: This is the length of the file after which the file will be truncated.

callback: This is the callback function no arguments other than a possible exception are given to the completion callback.

Program 5.8: Node JS program to demonstrate the fs.ftruncate() Method.

```
var fs = require("fs");
var buf = new Buffer.alloc(1024);

console.log("Going to open an existing file");
fs.open('programming.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to truncate the file after 10 bytes");

   // Truncate the opened file.
   fs.ftruncate(fd, 10, function(err) {
      if (err) {
         console.log(err);
      }
      console.log("File truncated successfully.");
   });
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Going to open an existing file
File opened successfully!
Going to truncate the file after 10 bytes
File truncated successfully.
```

5.2.6 Append a File

- It is used to asynchronously append the given data to a file using `appendFile()` method. If file does not exist this create new file.
- **Syntax:**

```
fs.appendFile( path, data[, options], callback )
```

Where,

path: It is source filename or file descriptor that will be appended to.

data: It is data that has to be appended.

options: It is an string or an object that can be used to specify optional parameters. It has three optional parameters:

- **encoding:** It is a string which specifies the encoding of the file. The default value is 'utf8'.
- **mode:** It is an integer which specifies the file mode. The default value is '0o666'.
- **flag:** It is a string which specifies the flag used while appending to the file. The default value is 'a'.
- **callback:** It is a function that would be called when the method is executed.
- **err:** It is an error that would be thrown if the method fails.

Program 5.9: Node JS program to demonstrate the fs.appendFile() method.

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

// Get the file contents before the append operation
console.log("\nFile before append:",
  fs.readFileSync("programming.txt", "utf8"));

fs.appendFile("programming.txt", "World", (err) => {
  if (err) {
    console.log(err);
```

```
    }
    else {
      // Get the file contents after the append operation
      console.log("\nFile after append:",
        fs.readFileSync("programming.txt", "utf8"));
    }
  });
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
File before append: This is a file containing a collection of programming
languages.
1. JAVA
2. C++
3. R
File after append: This is a file containing a collection of programming
languages.
1. JAVA
2. C++
3. R
World
```

5.3 OTHER I/O OPERATIONS

1. Stat( ):

- In Node, we can use the `stat()` method to get statistics about a file. The `stat()`, takes a path string and callback function as arguments.
- The callback function also takes two arguments. The first argument to the callback is the err object and the second is an object containing the stats found for the file.
- **Syntax:**

```
fs.stat(path, callback)
```

- `stat()` methods are given in the following table.

Table 5.3: stat() methods

Method	Description
`stats.isFile()`	It returns true if file type of a simple file.
`stats.isDirectory()`	It returns true if file type of a directory.

Contd..

stats.isBlockDevice()	It returns true if file type of a block device.
stats.isCharacterDevice()	It returns true if file type of a character device.
stats.isSymbolicLink()	It returns true if file type of a symbolic link.
stats.isFIFO()	It returns true if file type of a FIFO.
stats.isSocket()	It returns true if file type of a socket.

Program 5.10: Program for get statistics about file.

```
var fs = require("fs");
console.log("Get file info!");
fs.stat('programming.txt', function (err, stats) {
   if (err) {
      return console.error(err);
   }
   console.log(stats);
   console.log("Got file info successfully!");

   // Check file type
   console.log("isFile ? " + stats.isFile());
   console.log("isDirectory ? " + stats.isDirectory());
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Get file info!
Stats {
  dev: 3865741643,
  mode: 33206,
  nlink: 1,
  uid: 0,
  gid: 0,
  rdev: 0,
  blksize: 4096,
  ino: 844424930132133,
  size: 90,
  blocks: 0,
```

```
  atimeMs: 1608309396710.3035,
  mtimeMs: 1608309396643.2324,
  ctimeMs: 1608309396643.2324,
  birthtimeMs: 1608283886442.5571,
  atime: 2020-12-18T16:36:36.710Z,
  mtime: 2020-12-18T16:36:36.643Z,
  ctime: 2020-12-18T16:36:36.643Z,
  birthtime: 2020-12-18T09:31:26.443Z
}
Got file info successfully!
isFile ? true
isDirectory ? false
```

- Depending on the type of file or operating system you are using, the stats returned may differ than above. Below is a table that summarizes the stats that returned by the stat method.

Table 5.4: stats that returned by the stat method

Property	Description
dev	ID of the device containing the file.
mode	The file's protection.
nlink	The number of hard links to the file.
uid	User ID of the file's owner.
gid	Group ID of the file's owner.
rdev	The device ID, if the file is a special file.
blksize	The block size for file system I/O.
ino	The file's inode number. An inode is a file system data structure that stores information about a file.
size	The file's total size in bytes.
blocks	The number of blocks allocated for the file.
atime	Date object representing the file's last access time.
mtime	Date object representing the file's last modification time.
ctime	Date object representing the last time the file's inode was changed.

- There are more methods to check whether a path is pointing to a file or is a directory. The stats object returned from the callback function contains methods for finding this out:

```
var fs = require("fs");
var path = "C:\shiv\nodeexmaple\abcde.exe";
fs.stat(path, function(error, stats) {
    console.log(stats.isFile());
    console.log(stats.isDirectory());
    console.log(stats.isBlockDevice());
    console.log(stats.isCharacterDevice());
    console.log(stats.isFIFO());
    console.log(stats.isSocket());
});
```

2. Using Read():

- Use the `fs.read()` method to read file data. You must first use the open method before attempting to read from the file. The read method takes numerous arguments, but the most important ones to know are that the read method uses the file descriptor (obtained from the open method), along with the file size (obtained from stats).
- **Syntax:**

```
fs.read(fd, buffer, offset, length, position, callback)
```

Where,

fd: This is the file descriptor returned by fs.open().

buffer: This is the buffer that the data will be written to.

offset: This is the offset in the buffer to start writing at.

length: This is an integer specifying the number of bytes to read.

position: This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.

callback – This is the callback function which gets the three arguments, (err, bytesRead, buffer).

Program 5.11: Node JS program to demonstrate the fs.read() method.

```
const { Buffer } = require("buffer");
var fs = require("fs");
var buf = new Buffer.alloc(1024);

console.log("Open an existing file");
```

```
fs.open('programming.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Read the file");

   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
      if (err){
         console.log(err);
      }
      console.log(bytes + " bytes read");

      // Print only read bytes to avoid junk.
      if(bytes > 0){
         console.log(buf.slice(0, bytes).toString());
      }
   });
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Open an existing file
File opened successfully!
Read the file
90 bytes read
This is a file containing a collection of programming languages.
1. JAVA
2. C++
3. R
World
```

3. Using Close:

- To close a file, simply use the `close()` method and pass in the file descriptor (obtained from the open method).
- **Syntax:**

```
fs.close(fd, callback)
```

```
fetch('https://jsonplaceholder.typicode.com/MissingResource')
  .then(checkResponseStatus);
  .then(res => res.json());
  .then(json => console.log(json));
  .catch(err => console.log(err));
```

- We used the function at the beginning of the promise-chain (before parsing the response body) in order to see whether we encountered an issue. You can also throw a custom error instead.
- Again, you should have a strategy in place for handling errors like this instead of just printing to the console.
- If everything went as expected, and the status code indicated success, the program will proceed as before.

Summary

- The file system is big part of any application needing to handle files path for loading, manipulating or serving data.
- We can handle file operations such as create, read, delete and many more using Node JS. The built-in model provided by Node JS is `fs` (File System).
- The `fs.readFileSync()` method is an inbuilt application programming interface of `fs` module which is used to read the file and return its content.
- To read contents of a file into memory is very common programming task. There are many file system methods in `fs` module. Lets first start with `fs.readFile()`.
- If we want to write data synchronously we use `fs.writeFileSync()` method.
- If we want to write data asynchronously we use `fs.writeFile()` method.
- To open a file, create file, to write to a file or to read a file `fs.open()` method is used. `fs.open()` method performs several operations on a file.
- We want to delete a file we use `fs.unlink()` method. This method does not work on directories. To delete directories we have to use `fs.rmdir()`.
- The `fs.ftruncate()` method is used to change the size of the file i.e. either increase or decrease the file size.
- It is used to asynchronously append the given data to a file using `appendFile()` method. If file does not exist this create new file.
- In Node, we can use the `stat()` method to get statistics about a file.
- Use the `fs.read()` method to read file data.

Check Your Understanding

1. Which method of fs module is used to delete a file?
   (a) fs.delete(fd, len, callback) (b) fs.remove(fd, len, callback)
   (c) fs.unlink(path, callback) (d) None of the above.
2. Which of the following is the correct way to get an extension of a file?
   (a) fs.extname('main.js') (b) path.extname('main.js')
   (c) os.extname('main.js') (d) None of the above
3. A stream fires data event when there is data available to read.
   (a) false (b) true
4. What does the fs module stands for?
   (a) File Service (b) File System
   (c) File save (d) File Store
5. Which statement is valid in using a Node module fs in a Node based application?
   (a) Import fs (b) Package fs
   (c) var fs = require("fs") (d) var fs = import("fs")
6. Which of the following is true about readable stream?
   (a) Readable stream is used for read operation
   (b) Output of readable stream can be input to a writable stream
   (c) Both of the above
   (d) None of the above

ANSWER KEY

1. (c)	2. (b)	3. (b)	4. (b)	5. (d)
6. (c)				

Practice Questions

Q.I: Answer the following questions in short.

1. What is Synchronous and Asynchronous approach?
2. Write use of close()? – close a file
3. Write syntax and use of read().
4. Write use of stats function().
5. Why we use require method?

Q.II: Answer the following questions.

1. Explain with help of suitable example readFile() method.

2. Explain 2 ways to write in a file.
3. How to write synchronous data to a file explain with suitable example.
4. How to write asynchronous data to a file explain with suitable example.
5. Write a note on opening a file in Node JS.
6. How to delete file in Node.JS? Explain with suitable example.
7. Explain difference between blocking and non-blocking calls in Node JS.
8. Write a note on reading from file in Node JS.
9. Write short note on exception handling.

❖❖❖

fs.unlink()

6...

Events

Objectives...

- To learn about EventEmitter class.
- To understand Returning event emitter.
- To study Inhering events.

6.1 INTRODUCTION

- Every action in computer is event. Like button click, when a connection is made or a file is opened. Event driven programming is the programming paradigm in computer science field. Here flow of execution is determined by the events like user clicks or other programming threads or query result from database. Events are handled by event handlers or event callbacks.
- Most frameworks have an event driven model that allows for the setting and capturing of events though event triggers and listeners. We will learn about the methods that Node allows us to use for handling of events that should fit most of our application's event handling needs.

What is Event?

- An Event is an action that can be identified by a program related to hardware or software. Events can be user generated such as keystrokes, mouse click and system generated memory full, program loading, errors etc.

6.1.1 Event Driven Programming

- Node JS is very fast as compare to other similar technologies as uses events heavily. When Node starts its server it simply initiates its variables, declares functions and then simply waits for the event to occur.
- If we talk about an event driven application, there is generally a main loop that listens for events and then triggers a callback function when one of those events is detected.
- The working of event driven programming depends upon events.
- Events are monitored by a code (or function) known as an event listener.

- If the event listener detects that an assigned event has occurred, it will trigger a callback function, known as an event handler, which will perform event, e.g. clicking (the event) a "print" button (event listener) activates the actual print process (event handler).

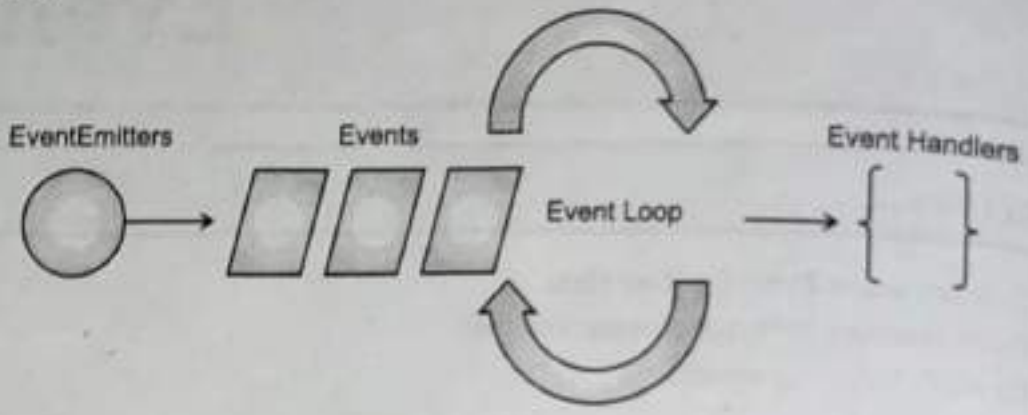


Fig. 6.1: Events in Node JS

- The difference between event and callback is callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern. The functions that listen to events act as **Observers**. Whenever an event gets invoked, its listener function starts executing.

```
// Import events module
var events = require('events');
// We will create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

- Following is the syntax to bind an event handler with an event:

```
// We will Bind event and event handler as below
eventEmitter.on('eventName', eventHandler);
```

- We can call an event programmatically as below,

```
// Call an event
eventEmitter.emit('eventName');
```

Program 6.1: Program for generate and invoke an event.

```
// Import events module
var events = require('events');

// create an eventEmitter object
var eventEmitter = new events.EventEmitter();
// Create an event handler as follows
```

```
var connectHandler = function connected() {
   console.log('connection has been successful.');
   // Fire the data_received event
   eventEmitter.emit('data_received');
}

// Bind the connection event with the handler
eventEmitter.on('connection', connectHandler);

// Bind the data_received event with the anonymous function
eventEmitter.on('data_received', function() {
   console.log('We have received data successfully.');
});

// Invoke the connection event
eventEmitter.emit('connection');

console.log("Program has Ended.");
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
connection has been successful.
We have received data successfully.
Program has Ended.
```

Program 6.2: Program for Listener and Binding.

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

// We have listener #1
var listner1 = function listner1() {
   console.log('We have executed  listner1.');
}

// We have listener #2
var listner2 = function listner2() {
```

```
    console.log('We have executed listner2');
}

// We will bind the connection event with the listner1 function
eventEmitter.addListener('connection', listner1);

// We will bind the connection event with the listner2 function
eventEmitter.on('connection', listner2);

var eventListeners = require('events').EventEmitter.listenerCount
(eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");

// We will invoke the connection event
eventEmitter.emit('connection');

// We will remove the binding of listner1 function
eventEmitter.removeListener('connection', listner1);
console.log("Now Listner1 will not listen.");

// We will invoke the connection event
eventEmitter.emit('connection');

eventListeners=require('events').EventEmitter.listenerCount(eventEmitter,
connection');
console.log(eventListeners + " Listner(s) listening to connection event");

console.log("Our Program has been Ended.");
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
2 Listner(s) listening to connection event
We have executed  listner1.
We have executed listner2
Now Listner1 will not listen.
```



```
We have executed listner2
1 Listner(s) listening to connect
Our Program has been Ended.
PS D:\Prajakta\BBA CA\Node JS\pro
```

6.2 EVENTEMMITER CLASS

- Node JS allows you to create and han
- Event module includes EventEmitt and handle custom events. Lots of ob
- Suppose if we consider as an exam connects to it; a fs.readStream em which emit events are the instance
- Node JS has multiple in-built EventEmitter class.
- EventEmitter class can be accessib

```
// First import events mod
var events = require('even
// second create an eventE
var eventEmitter = new eve
```

- If there is any error then EventE added, 'newListener' event is cal event is called.
- EventEmitter gives us multiple bind a function with the event a

Methods of EventEmitter class:

Table 6.1:

Sr. No.	Method
1.	addListener(event, list
2.	on(event, listener)
3.	once(event, listener)

```
We have executed listner2
1 Listner(s) listening to connection event
Our Program has been Ended.
PS D:\Prajakta\BBA CA\Node JS\programs>
```

6.2 EVENTEMMITER CLASS

- Node JS allows you to create and handle custom events easily by using events module.
- Event module includes EventEmitter class. EventEmitter class can be used to raise and handle custom events. Lots of objects in a Node emit events.
- Suppose if we consider as an example, a net.Server emits an event each time a peer connects to it; a `fs.readStream` emits an event when the file is opened. All objects which emit events are the instances of `events.EventEmitter`.
- Node JS has multiple in-built events available through events module and EventEmitter class.
- EventEmitter class can be accessible by using the following code:

```
// First import events module
var events = require('events');
// second create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

- If there is any error then EventEmitter emits an 'error' event. When a new listener is added, `'newListener'` event is called and when a listener is removed, `'removeListener'` event is called.
- EventEmitter gives us multiple properties like **on** and **emit**. The **on** property is used to bind a function with the event and **emit** is used to call the event.

Methods of EventEmitter class:

Table 6.1: Methods of EventEmitter Class

Sr. No.	Method	Description
1.	`addListener(event, listener)`	It is used to add a listener at the end of the listeners array for the specified event.
2.	`on(event, listener)`	It is used to add a listener at the end of the listeners array for the specified event.
3.	`once(event, listener)`	It is used to add a onetime listener to the event. This listener is called only the next time the event is fired, after which it is removed.

4.	`removeListener(event,listener)`	It is used to removes a listener from the listener array for the specified event.
5.	`removeAllListeners([event])`	It is used to removes all listeners, or those of the specified event.
6.	`setMaxListeners(n)`	This method will by default, EventEmitters will print a warning if more than 10 listeners are added for a particular event.
7.	`defaultMaxListeners()`	It is used to change the default value for all EventEmitter instances, the EventEmitter.defaultMaxListeners property can be used.
8.	`getMaxListeners()`	The EventEmitter.getMaxListeners() will return the max listeners value set by setMaxListeners() or default value 10.
9.	`listeners(event)`	It is used to return an array of listeners for the specified event.
10.	`emit(event,[arg1],[arg2],[...])`	It is used to execute each of the listeners in order with the supplied arguments.
11.	`listenerCount(emitter, event)`	It is used to return the number of listeners for a given event.

Events:

Table 6.2: Events

Sr. No.	Events and Description
1.	`newListener` • **event** – It is String: the event name. • **listener** – It is Function: the event handler function. This event is emitted any time a listener is added.
2.	`removeListener` • **event** – It is String, the event name. • **listener** – It is Function, the event handler function. This event is emitted any time when someone removes a listener.

Program 6.3: Program for Simple Event.

```
// Import events
const EventEmitter = require('events');

// Initializing event emitter instances
var eventEmitter = new EventEmitter();

// Registering to myEvent
eventEmitter.on('myEvent', (msg) => {
  console.log(msg);
});

// Triggering myEvent
eventEmitter.emit('myEvent', "First event");
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
First event
```

Program 6.4: Program for removing Listener.

```
const EventEmitter = require('events');
var eventEmitter = new EventEmitter();
var fun1 = (msg) => {
  console.log("Message from fun1: " + msg);
};

var fun2 = (msg) => {
  console.log("Message from fun2: " + msg);
};
// Registering fun1 and fun2
eventEmitter.on('myEvent', fun1);
eventEmitter.on('myEvent', fun1);
eventEmitter.on('myEvent', fun2);
// Removing listener fun1
eventEmitter.removeListener('myEvent', fun1);
// Triggering myEvent
```

```
eventEmitter.emit('myEvent', "Event occurred");
// Removing all the listeners to myEvent
eventEmitter.removeAllListeners('myEvent');
// Triggering myEvent
eventEmitter.emit('myEvent', "Event occurred");
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Message from fun1: Event occurred
Message from fun2: Event occurred
```

- We registered, two times fun1 and one time fun2 for calling eventEmitter.removeListener ('myEvent', fun1) one instance of fun1 will be removed. Finally, removing all listener by removeAllListeners() will remove all listeners to myEvent.

Program 6.5: Program for getMaxListener(), setMaxListener(), defaultMaxListener.

```
const EventEmitter = require('events');
// Initializing event emitter instances
var eventEmitter1 = new EventEmitter();
var eventEmitter2 = new EventEmitter();

// Getting max listener
console.log("Default max listener for eventEmitter1 is: ",
            eventEmitter1.getMaxListeners());
console.log("Default max listener for eventEmitter2 is: ",
            eventEmitter2.getMaxListeners());

// Set global deaultMaxListeners to 2
EventEmitter.defaultMaxListeners = 2;

// Getting max listener
console.log("Default max listener for eventEmitter1 is: ",
            eventEmitter1.getMaxListeners());
console.log("Default max listener for eventEmitter2 is: ",
            eventEmitter2.getMaxListeners());

// Set max listener of eventEmitter1 to 5
```

```
eventEmitter1.setMaxListeners(5);
// Getting max listener
console.log("Default max listener for eventEmitter1 is: ",
            eventEmitter1.getMaxListeners());
console.log("Default max listener for eventEmitter2 is: ",
            eventEmitter2.getMaxListeners());
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Default max listener for eventEmitter1 is:  10
Default max listener for eventEmitter2 is:  10
Default max listener for eventEmitter1 is:  2
Default max listener for eventEmitter2 is:  2
Default max listener for eventEmitter1 is:  5
Default max listener for eventEmitter2 is:  2
```

Program 6.6: Use of EventEmitter & Listener.

```
// Imports events
const EventEmitter = require('events');

// Initialize event emitter instances
var eventEmitter = new EventEmitter();

// declare listener fun1 to myEvent1
var fun1 = (msg) => {
    console.log("Message from fun1: " + msg);
};

// Declare listener fun2 to myEvent2
var fun2 = (msg) => {
    console.log("Message from fun2: " + msg);
};

// Listen to myEvent with fun1 and fun2
eventEmitter.addListener('myEvent', fun1);
```

```
// fun2 will be inserted in front of listeners array
eventEmitter.prependListener('myEvent', fun2);

// Listing listeners
console.log(eventEmitter.listeners('myEvent'));

// Count the listeners registered to myEvent
console.log(eventEmitter.listenerCount('myEvent'));

// Triggering myEvent
eventEmitter.emit('myEvent', 'Event occurred');
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
[ [Function: fun2], [Function: fun1] ]
2
Message from fun2: Event occurred
Message from fun1: Event occurred
```

6.3 RETURNING EVENTEMITTER

- When we are returning from EventEmitter we use constructor function which returns EventEmitter object.
- This EventEmitter object can be used to subscribe for the events. Consider the following example.

Program 6.7: Program for returning eventemitter.

```
var emitter = require('events').EventEmitter;

function LoopProcessor(num) {
    var e = new emitter();
    setTimeout(function () {
        for (var i = 1; i <= num; i++) {
            e.emit('BeforeProcess', i);
            console.log('Processing number:' + i);
            e.emit('AfterProcess', i);
        }
    }
```

```
    , 2000)
    return e;
}
var lp = LoopProcessor(3);

lp.on('BeforeProcess', function (data) {
    console.log('About to start the process for ' + data);
});

lp.on('AfterProcess', function (data) {
    console.log('Completed processing ' + data);
});
```

Output:

```
About to start the process for 1
Processing number:1
Completed processing 1
About to start the process for 2
Processing number:2
Completed processing 2
About to start the process for 3
Processing number:3
Completed processing 3
```

- In the above `LoopProcessor()` function, first we create an object of `EventEmitter` class and then use it to emit `'BeforeProcess'` and `'AfterProcess'` events. Finally, we return an object of `EventEmitter` from the function. So now, we can use the return value of `LoopProcessor` function to bind these events using `on()` or `addListener()` function.

6.4 INHERITING EVENTS

- We can use EventEmitter Class to inherit events. For that we have to extend the constructor function from EventEmitter class to emit the events.
- We will be extending `LoopProcessor` constructor function with `EventEmitter` class using `util.inherits()` method of utility module. So, we can use `EventEmitter`'s methods with `LoopProcessor` object to handle its own events.

Program 6.8: Program to Extend EventEmitter Class.

```
var emitter = require('events').EventEmitter;
var util = require('util');

function LoopProcessor(num) {
    var me = this;
    setTimeout(function () {
        for (var i = 1; i <= num; i++) {
            me.emit('BeforeProcess', i);
            console.log('Processing number:' + i);
            me.emit('AfterProcess', i);
        }
    }
    , 2000)
    return this;
}
util.inherits(LoopProcessor, emitter)
var lp = new LoopProcessor(3);
lp.on('BeforeProcess', function (data) {
    console.log('About to start the process for ' + data);
});
lp.on('AfterProcess', function (data) {
    console.log('Completed processing ' + data);
});
```

Output:

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
About to start the process for 1
Processing number:1
Completed processing 1
About to start the process for 2
Processing number:2
Completed processing 2
About to start the process for 3
Processing number:3
Completed processing 3
PS D:\Prajakta\BBA CA\Node JS\programs>
```

Summary

- Every action in computer is event. Like button click, when a connection is made or a file is opened.
- An Event is an action that can be identified by a program related to hardware or software. Events can be user generated such as keystrokes, mouse click and system generated memory full, program loading, errors etc.
- Node JS allows you to create and handle custom events easily by using events module.
- Event module includes EventEmitter class. EventEmitter class can be used to raise and handle custom events.
- When a new listener is added, `'newListener'` event is called and when a listener is removed, `'removeListener'` event is called.
- EventEmitter gives us multiple properties like **on** and **emit**. The **on** property is used to bind a function with the event and **emit** is used to call the event.
- We can use EventEmitter Class to inherit events. For that we have to extend the constructor function from EventEmitter class to emit the events.

Check Your Understanding

1. Which of the following provides in-built events?
   (a) events (b) callback
   (c) throw (d) handler
2. Which of the following class is used to create and consume custom events in Node JS?
   (a) EventEmitter (b) Events
   (c) NodeEvent (d) None of the above
3. Which function is used to include file modules in Node JS?
   (a) include() (b) require()
   (c) attach() (d) None of the above
4. Node JS uses event-driven non – blocking I/O model?
   (a) True (b) False
5. Something that happened in out application that we can respond to____.
   (a) Events (b) Actions
   (c) Procedures (d) Callback
6. What is used to returning from EventEmitter object?
   (a) Get() function (b) Set() Function
   (c) Constructor Function (d) User function

7. Which class is use to inherit events?
   (a) Event (b) Listner
   (c) EventEmitter (d) Emitter

ANSWER KEY

1. (a)	2. (a)	3. (b)	4. (a)	5. (a)
6. (c)	7. (c)			

Practice Questions

Q.I: Answer the following questions in short.

1. What is the use of on and emit property?
2. Write difference between removeListener() and removeAllListener()?
3. What is function of eventListener?
4. How to extend EventEmmitter class?
5. How to return EventEmmitter from function?

Q.II: Answer the following questions.

1. What is event driven programming?
2. Explain EventEmmiter Class.
3. Explain methods of EventEmmiter Class. Write a program which illustrates the use of EventEmmiter Class.
4. How we return event emitter? Explain with suitable example.
5. With the help of suitable example explain Inheriting events.

Q.III: Define the terms.

1. EventEmmitter
2. removeListener
3. newListener
4. Event

❖❖❖

7...

Database Connectivity

Objectives...

- To establishing connection with database.
- To creating database through MySQL dashboard and query.
- To creating Schema / Tables.
- To insert records and display records from table.
- To update and delete records from table.

7.1 INTRODUCTION

- A Database connection is a facility in computer science that allows client software to talk to database server software, whether on the same machine or not. A connection is required to send commands and receive answers, usually in the form of a result set.

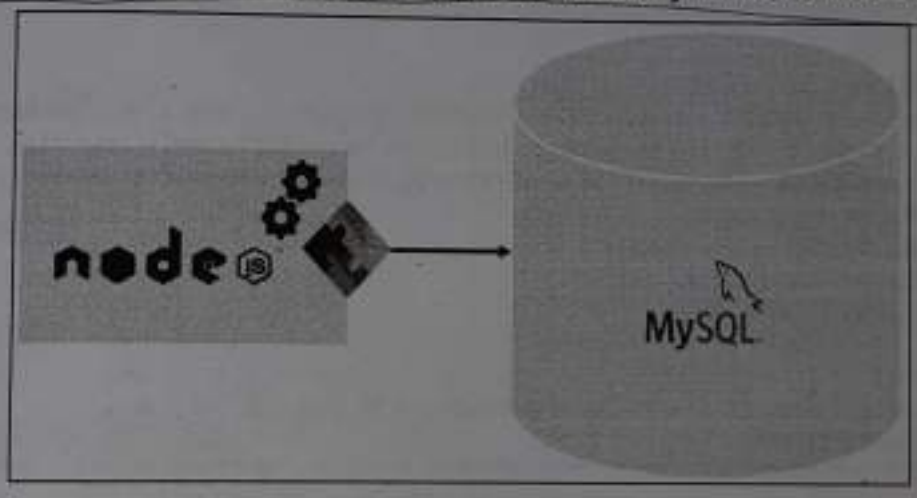


Fig. 7.1: Front End and Back End

- Connections are built by supplying an underlying driver or provider with a connection string, which is a way of addressing a specific database or server.

- Database connectivity with MySQL server using NODE JS is quite simple. We simply need to connect to the MySQL database server from a Node. Make sure that your system is having MySQL installed. If it is not, then do download the appropriate version from the link given below:

  `https://www.mysql.com/downloads/`

  OR

  `https://dev.mysql.com/get/Downloads/MySQLInstaller/mysql-installer-community-8.0.19.0.msi`

- If your system is installed with WampServer or Xampp Sever, then it is also okay for us to have database connectivity with MySQL server. Because both these servers are having MySQL built-in available in their set up as one of the services.

7.2 CONNECTION STRING

Follow the steps given below for Database Connectivity:

Step 1: Create a project folder with valid and meaningful name under any available drives. For example: We create one folder with name **BCA** under **C drive**.

Step 2: Install MySQL Driver. To access a MySQL database with Node JS, we need a MySQL driver.

We will be using the "mysql" module, downloaded from NPM(Node Package Manager).

Now, go to command prompt and execute the following command:

```
C:\Users\UserName>npm install mysql
```

When you execute above command, you may see following details. It shows that you have successfully installed MySQL database driver.

```
C:\Users\DELL>npm install mysql
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\DELL\package.json'
npm [redacted] created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\DELL\package.json'
npm WARN DELL No description
npm WARN DELL No repository field.
npm WARN DELL No README data
npm WARN DELL No license field.

+ mysql@2.18.1
added 11 packages from 15 contributors and audited 11 packages in 2.545s
found 0 vulnerabilities
```

Fig. 7.2 : Installation of MySQL driver for Node JS

Step 3: Creating a Connection: Now, we should write a code to connect with the MySQL database server. Write following code in a notepad or any editor and save it under **BCA** folder with the name **db_connect.js**

Program 7.1: Program for creating database connection.

db_connect.js

```
// include mysql module
var mysql = require(mysql);

// create a connection variable with the required details
var con = mysql createConnection({
   host: "localhost",          // IP address of server running mysql
   user: "root",               // user name to your mysql database
   password: "iccs#123",       // corresponding password.
   port : '3308'               /* Optional. But mention it if multiple
                               servers(such as MariaDB, MySQL) are running
                               simultaneously */
});

// make connection to the database
con.connect(function(err) {
   if (err) throw err;
   // if connection is successful
   console.log("Connected to the MySQL server!");
});

//Closing the connection
con.end(function(err) {
   if (err) {
         return console.log('error:' + err.message);
   }
   console.log('Closed the active database connection.');
});
```

Output:

```
C:\BCA>node db_connect.js
Connected to the MySQL server!
```

Step 4: Now, as the connection is established successfully with MySQL server database, we can start querying the database using SQL statements.

7.3 CONFIGURING

7.3.1 Creating Database in MySQL Server

A. Method : 1(Using Dashboard of MySQL Server)

- To create a new database using the MySQL Workbench, follow these simple steps:

1. **First, launch the MySQL Workbench.**

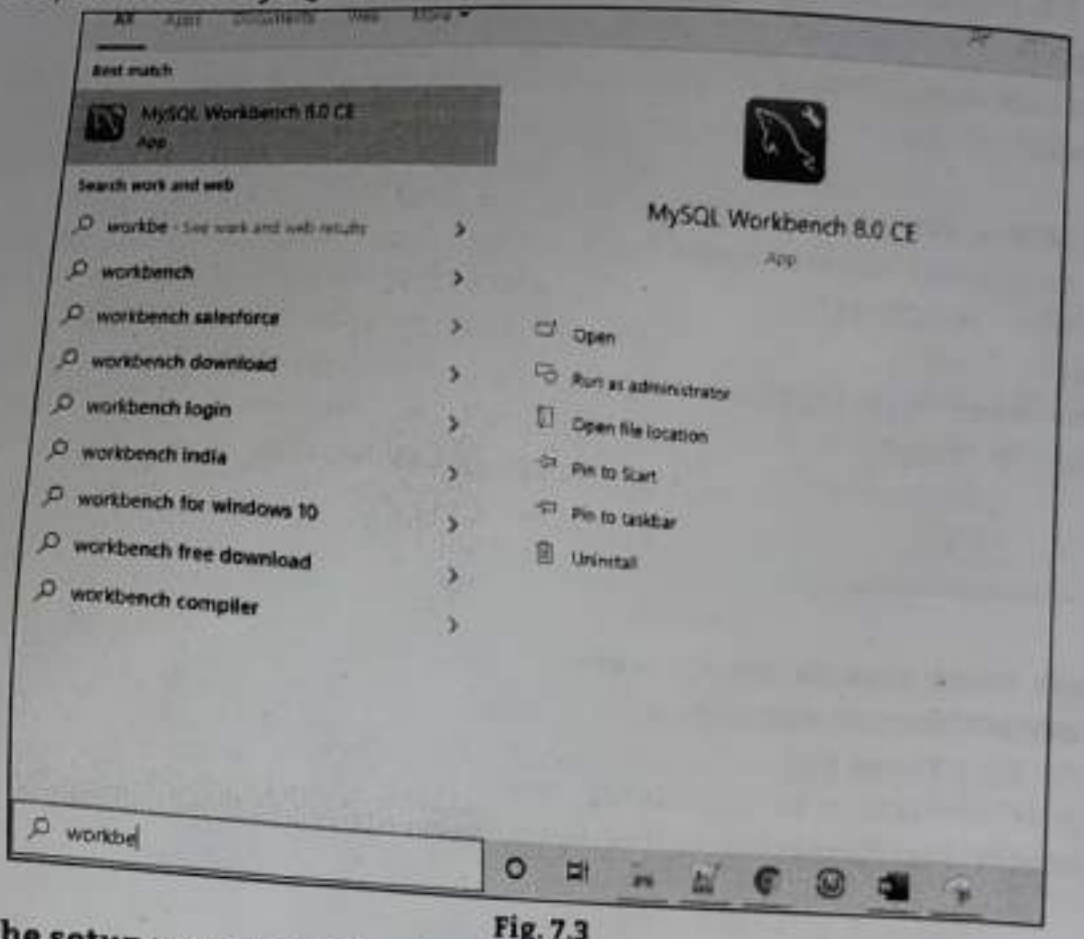


Fig. 7.3

2. **Click the setup new connection button as shown in the following screenshot.**

Fig. 7.4

3. **Type the name for the connection and click the Test Connection button.**

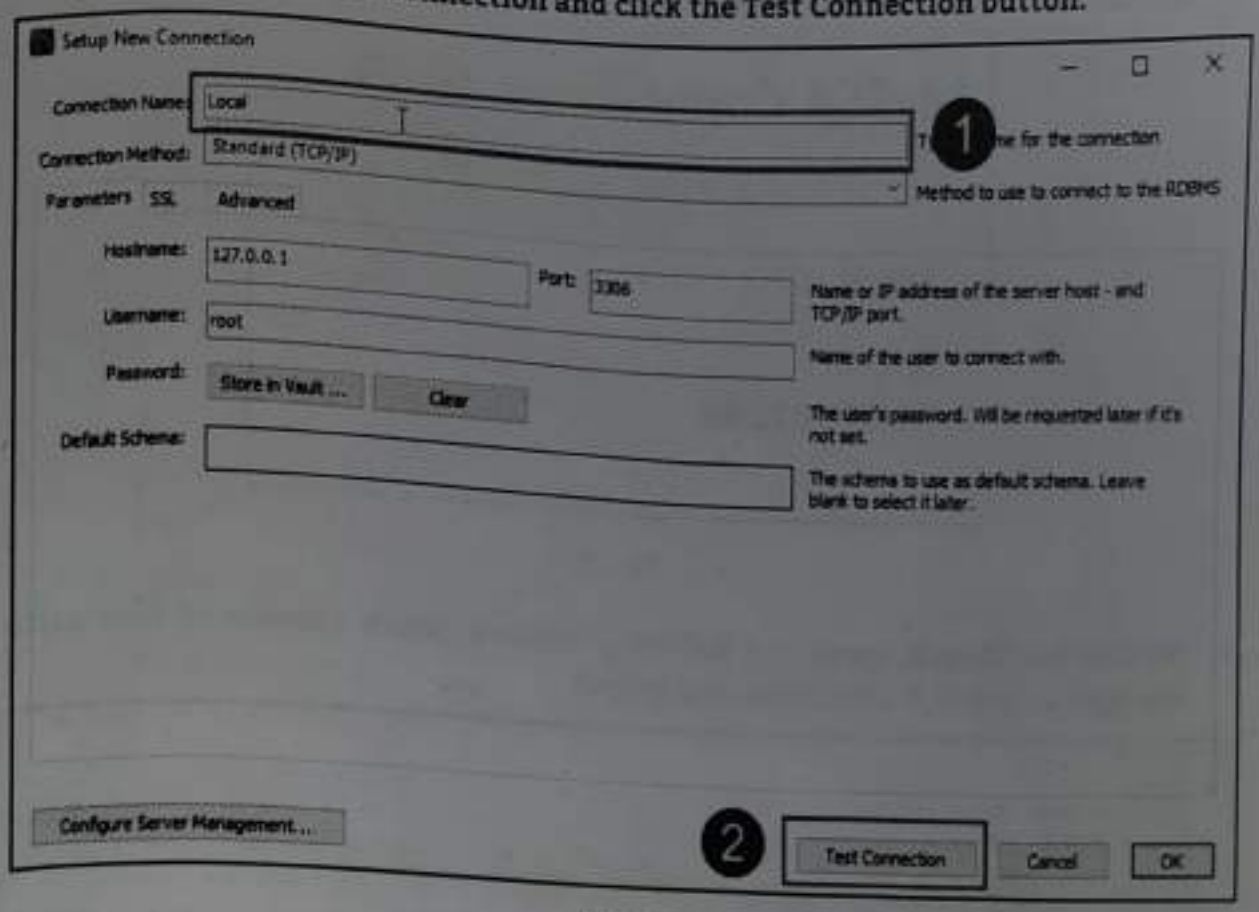


Fig. 7.5

4. **MySQL Workbench displays a dialog asking for the password of the root user.**

Fig. 7.6

You need to type the password for the root user, then check the Save password in vault, and click OK button.

5. **Double click the connection name Local to connect to the MySQL Server.**

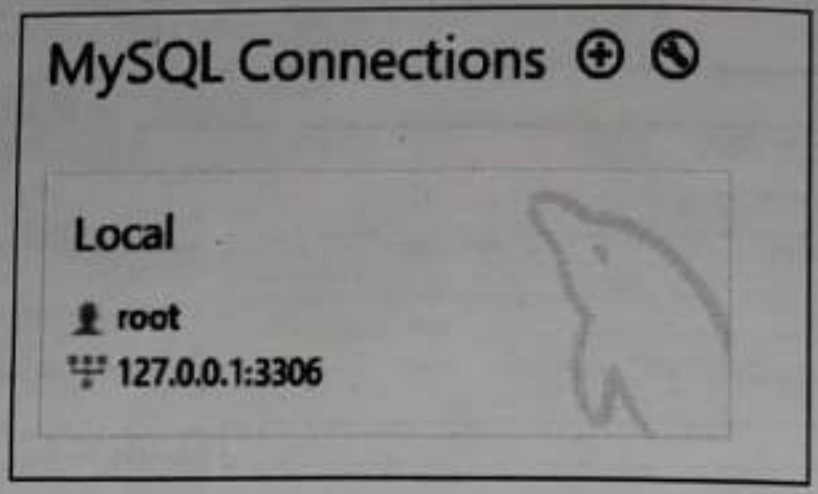


Fig. 7.7

6. **MySQL Workbench opens the following window which consists of four parts: Navigator, Query, Information, and Output.**

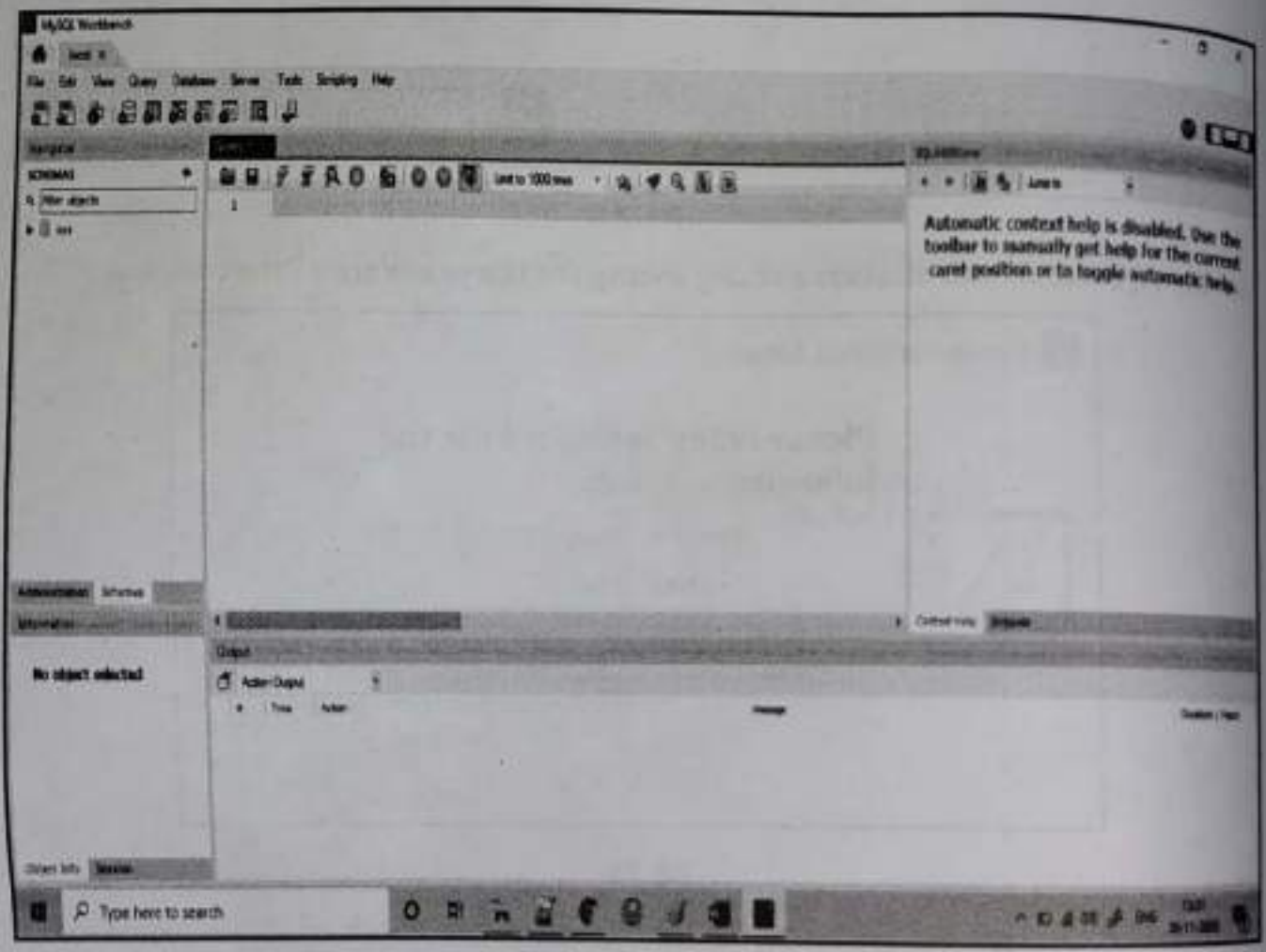


Fig. 7.8

7. **Click the create a new schema in the connected server button from the toolbar.**

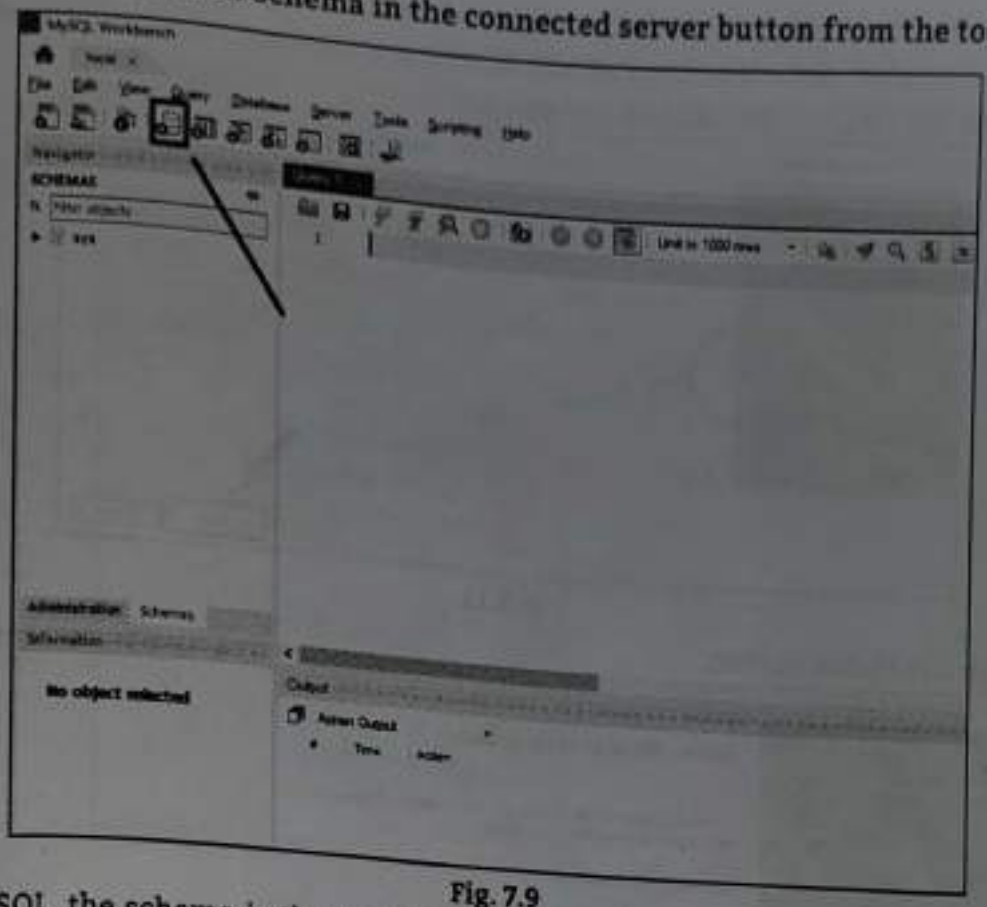


Fig. 7.9

- In MySQL, the schema is the synonym for the database. Creating a new schema also means creating a new database.

8. **The following window is open. You need to enter the schema name, then change the character set and collation if necessary, and click the Apply button.**

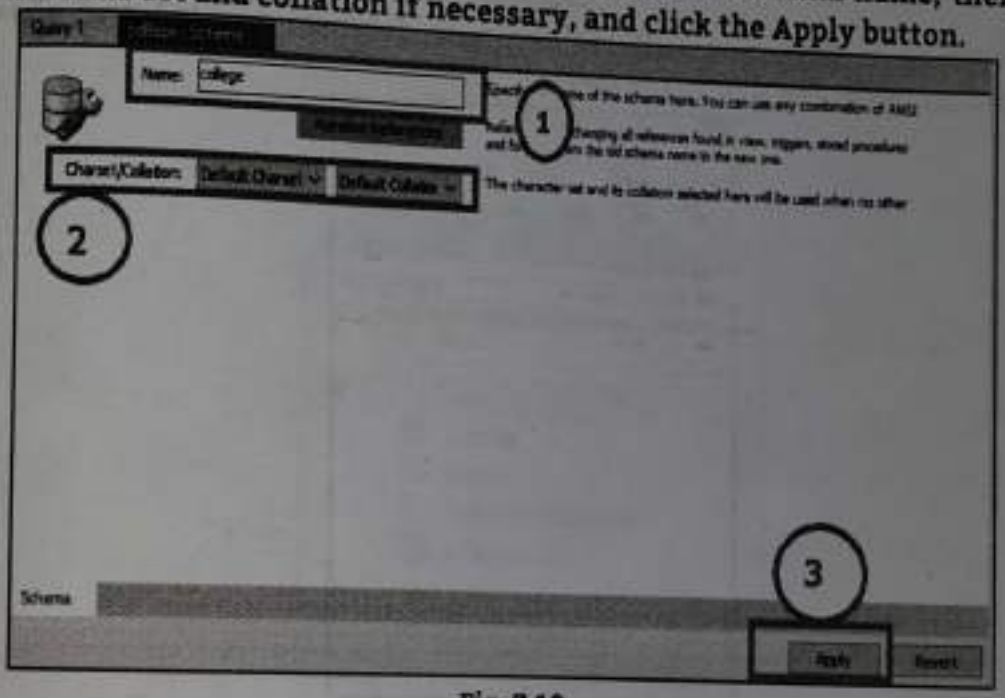


Fig. 7.10

9. MySQL Workbench opens the following window that displays the SQL script which will be executed. Note that the CREATE SCHEMA statement command has the same effect as the CREATE DATABASE statement.

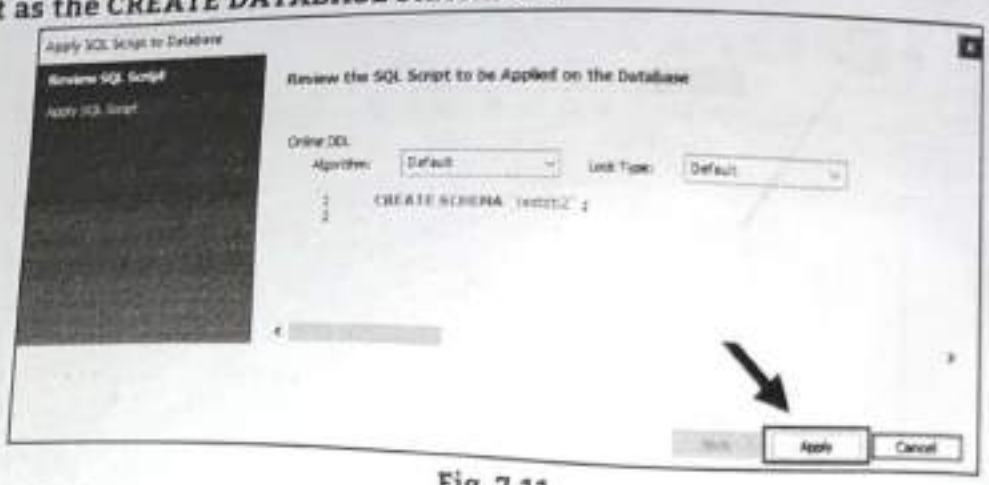


Fig. 7.11

10. Click on the Finish button.

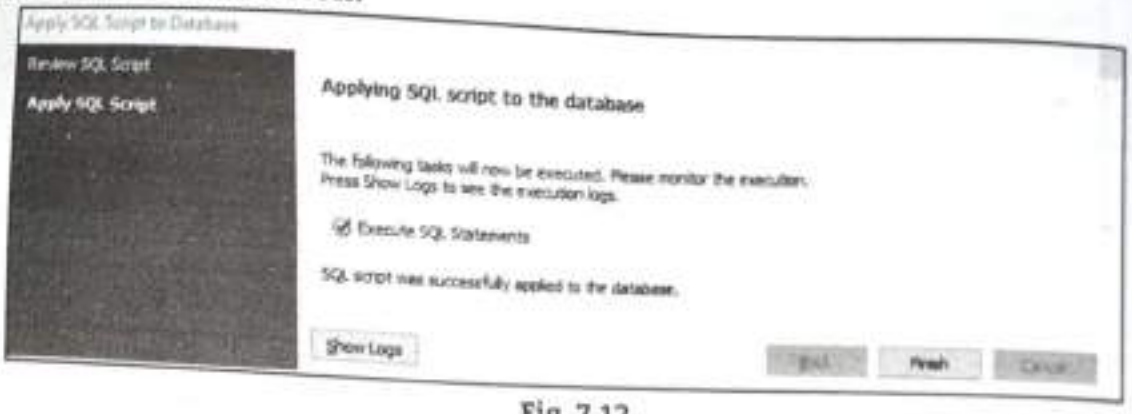


Fig. 7.12

11. If everything is fine, you will see the new database created and showed in the schemas tab of the Navigator section.

Fig. 7.13

12. To select the college database then right click the database name and choose Set as Default Schema menu item.

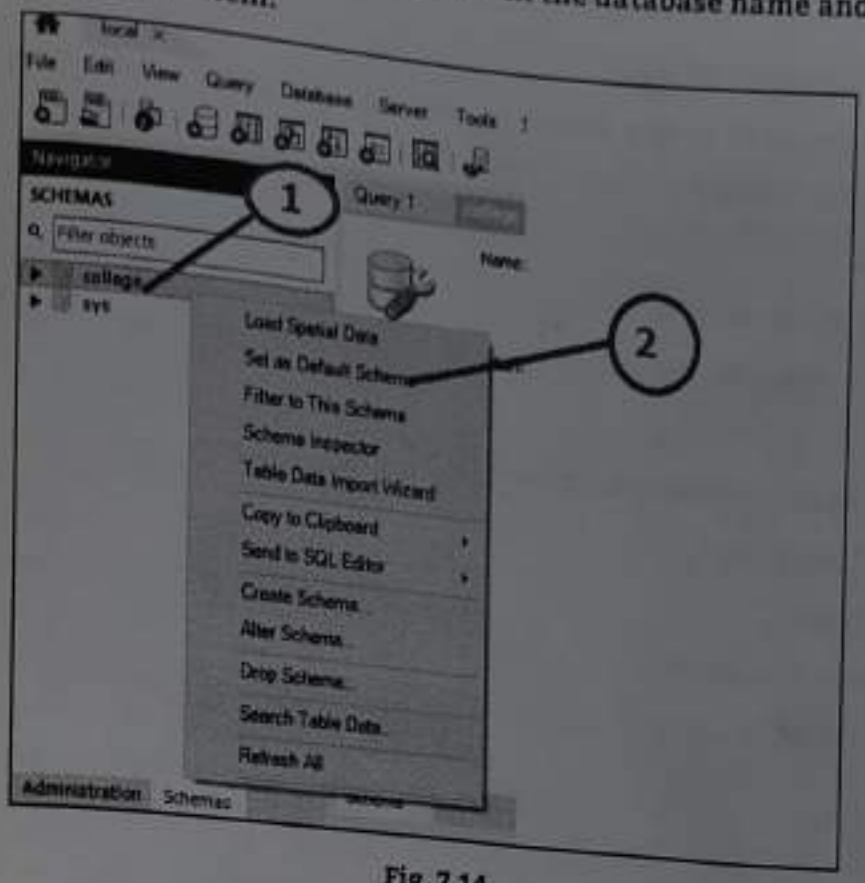


Fig. 7.14

13. The college node is open as shown in the following figure.

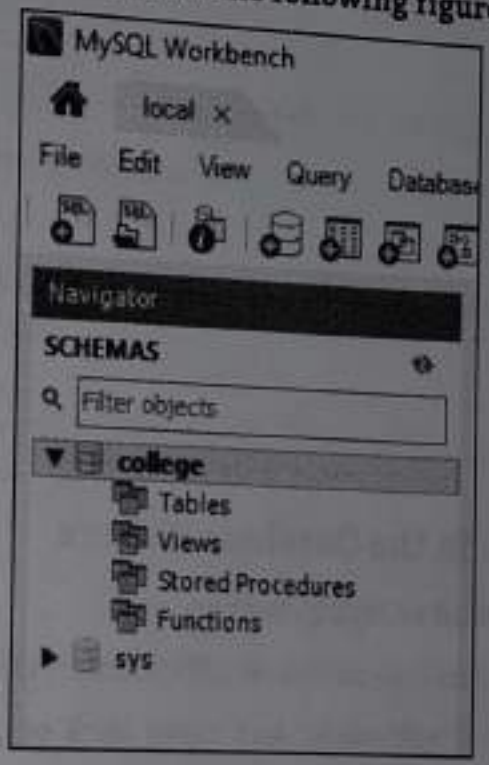


Fig. 7.15

14. **Now, we can work with college from the MySQL Workbench.**

B. Method : 2(Through Coding)

- Write following code in any editor, save it with the name **create_db.js** and execute this file from command prompt.

create_db.js

```
// include mysql module
var mysql = require('mysql');

var con = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "iccs#123",
   port : '3308'
});

//Making connection to database
con.connect(function(err) {
   if (err) throw err;
   console.log("Connected to the Database!");
   con.query("CREATE DATABASE college", function (err, result) {
      if (err) throw err;
      console.log("Database with name college created succssfully...");
   });
});
```

7.3.2 Creating a Table in the Database College

A. Method : 1(Using Dashboard of MySQL Server)

- To create new table under college database, follow simple steps given below.

1. **Open the MySQL Server Workbench. And right click on the database name then click on Create table option.**

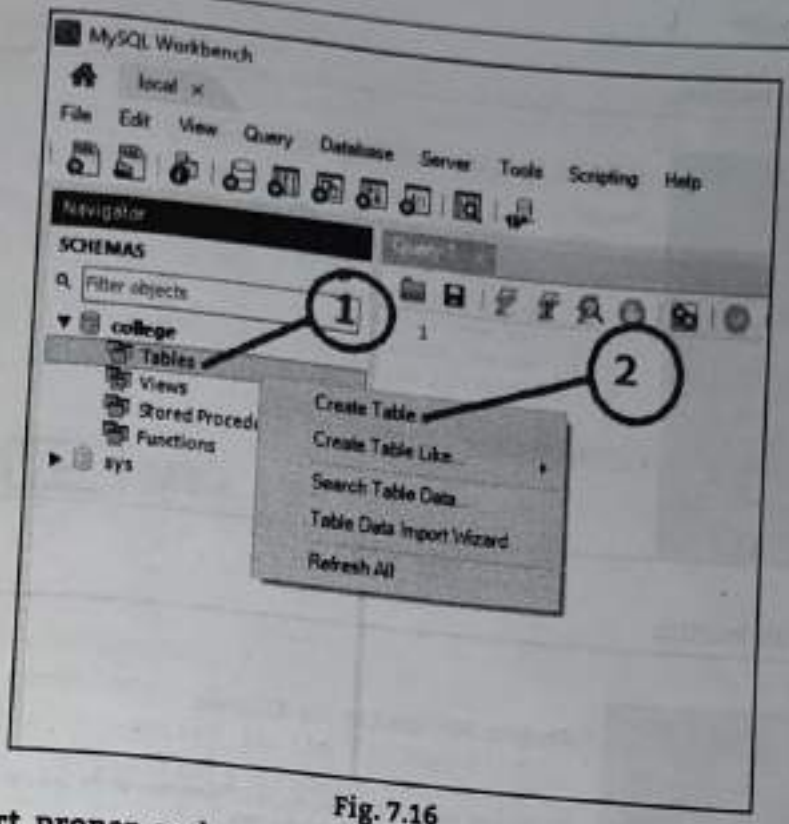


Fig. 7.16

2. **Now, 1st insert proper and meaningful name to the table. 2nd Create necessary fields with appropriate names and also select their type (datatype) then 3rd Click on Apply.**

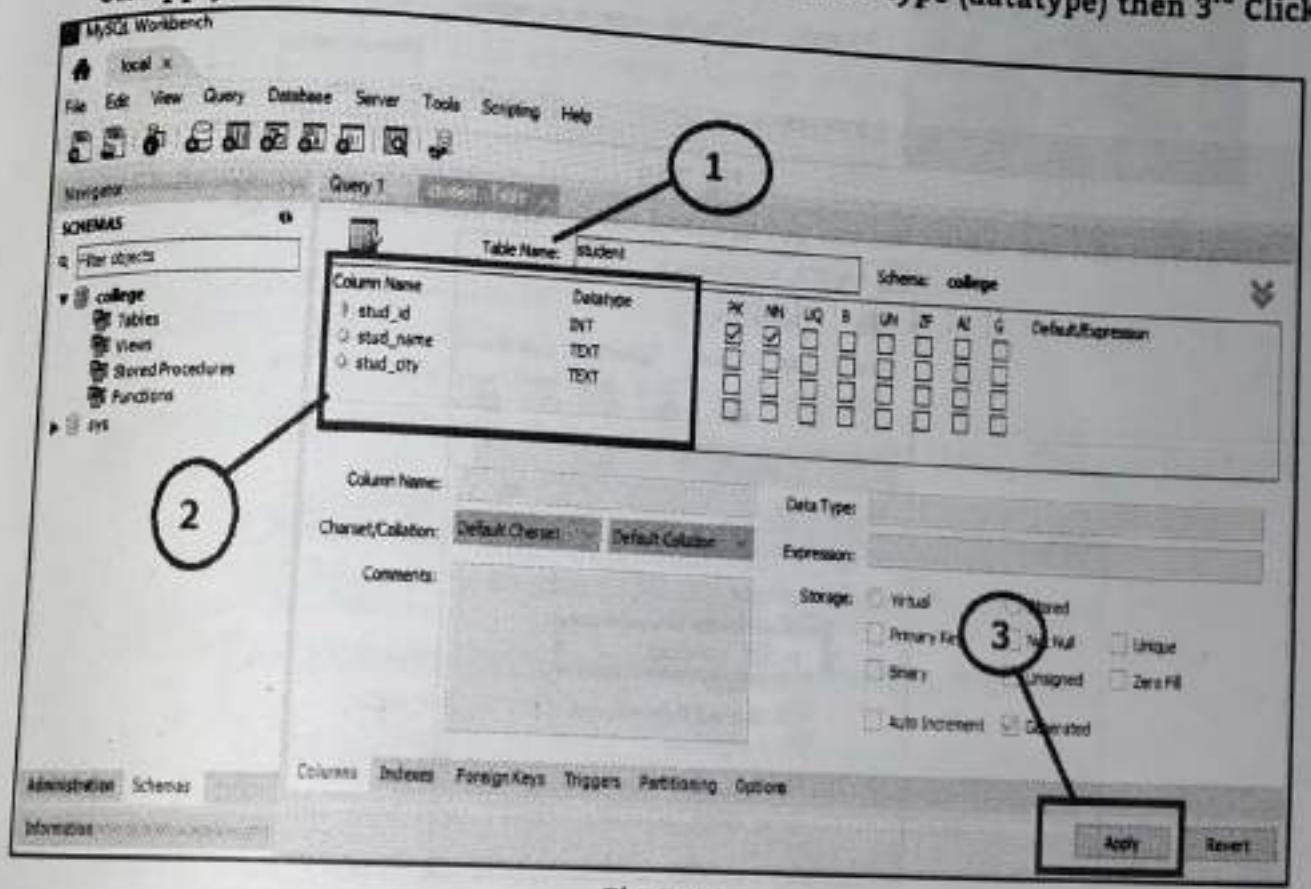


Fig. 7.17

3. Click on Apply button.

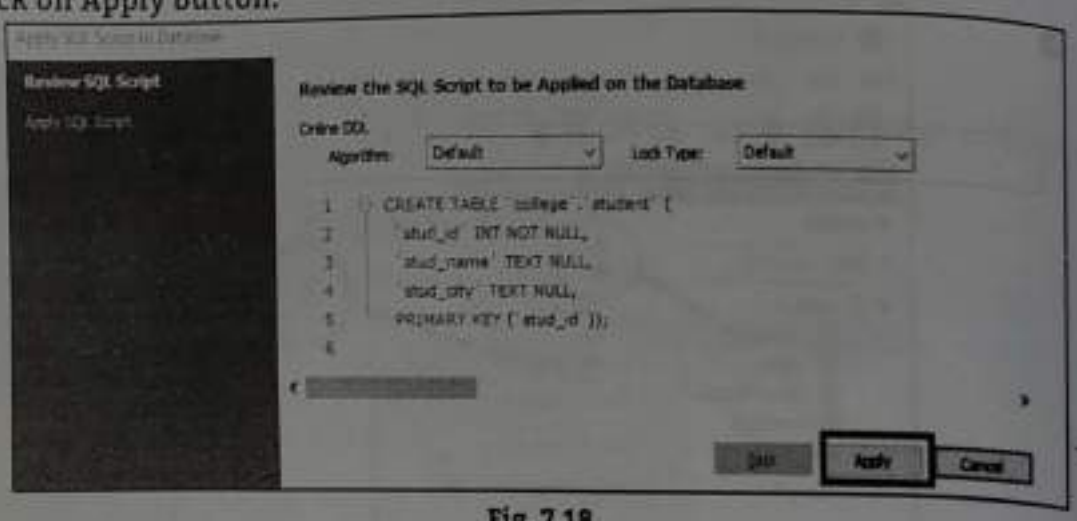


Fig. 7.18

4. Click on Finish button

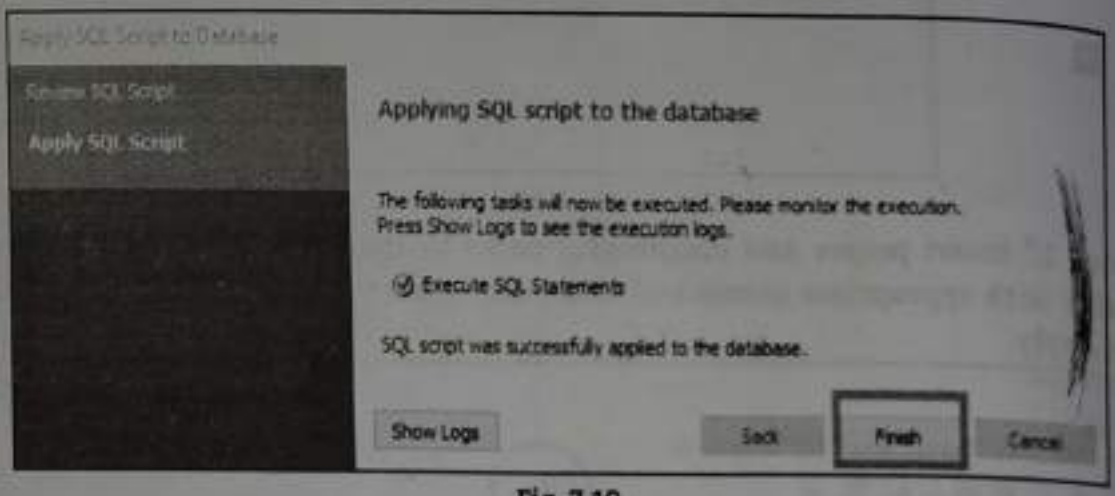


Fig. 7.19

5. Now, you can see table student got created under the database college.

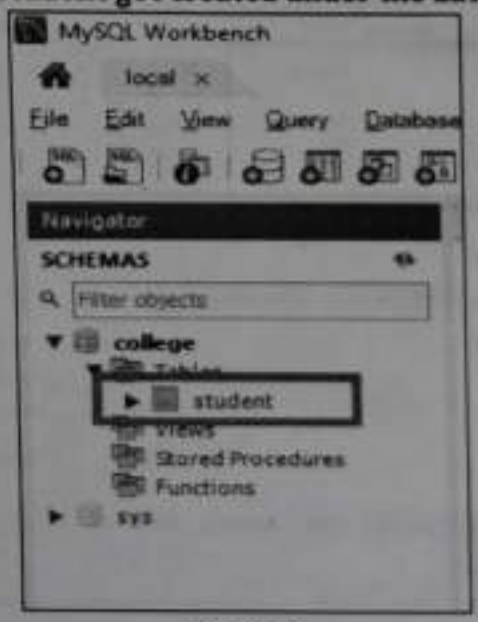


Fig. 7.20

B. Method : 2 (Through Coding)

- Here, we will be using "CREATE TABLE" statement of MySQL. Now, we need to specify the database name in which we are going to create a table.

Program 7.1: Program to create table student.

create_table.js

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "iccs#123",
  database: "college",
  port : '3308'
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected to the Database!");
  var sql = "CREATE TABLE student(stud_roll int, stud_name text, stud_city
  text, PRIMARY KEY(stud_roll))";

  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table with name student created successfully...");
  });
});
```

Output:

```
C:\BCA>node create_table.js
Connected to the Database!
Table with name student created successfully...
```

- You can see the View/Structure of the student table in the MySQL dashboard as shown below:

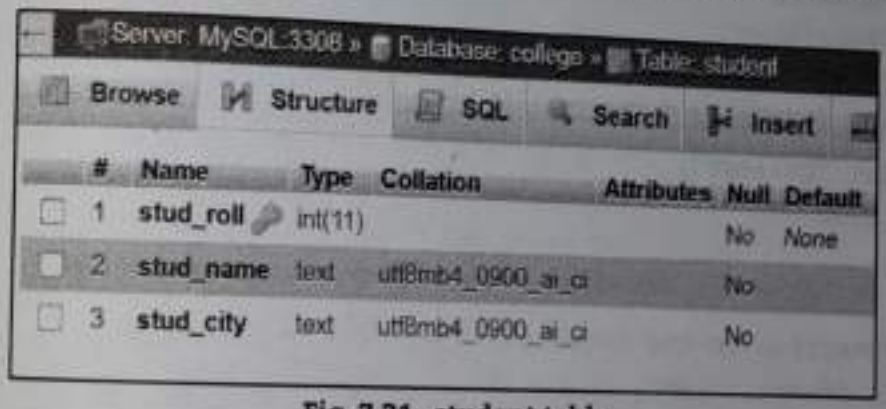


Fig. 7.21: student table

7.3.3 Insert a Record in the Table

Insert Query:

- The figure given below is the schema of table named **student** available under the database **college**. This table contains 3 columns namely **stud_roll, stud_name** and **stud_city**.

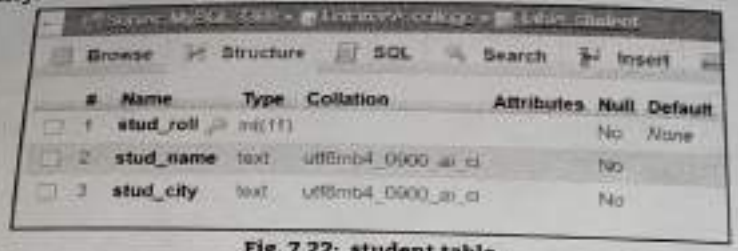


Fig. 7.22: student table

- *Observe that, the field named* **stud_roll** *is having type* **int** *and is a primary key of this table student. Other two fields namely* **stud_name** *and* **stud_city** *are of the type* **text**. *None of the field is null. All fields are made compulsory.*
- *Now, we are going to write* **insert** *query against this table* **student**. *We need to provide 3 values to the query.*

Program 7.2: Program to insert record in table student.

insert_query.js

```
//include Prompt-sync module
const prompt = require ('prompt-sync') ();

// include mysql module
var mysql = require('mysql');

// create a connection variable with the required details
var con = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "Iccs#123",
   database: "college",
   port : 3308
});

// make connection to the database
con.connect (function(err) {
   if (err) throw err;
```



```
// if connection is successful
console.log(Connected!');

//Accepting the user input from t
const roll = prompt('Enter Roll N
const name = prompt('Enter Stude
const city = prompt('Enter City

//Designing Dynamic Query based
var sql = "INSERT INTO student
("+ Number(roll) + ", '" + name

con.query(sql, function (err,
   if (err) throw err;
   // if there is no error,
   console.log("1 record suc
});
});
```

Output:

```
C:\BCA>node insert_query.js
Connected!
Enter Roll Number :101
Enter Student Name :Janard
Enter City :Pune
1 record successfully inse
```

- Assume that we have entered t
  below shows the records curren

stud_roll
101
102
10
10
1
1

```
    // if connection is successful
    console.log(Connected!');

    //Accepting the user Input from the user
    const roll = prompt('Enter Roll Number :');
    const name = prompt('Enter Student Name :');
    const city = prompt('Enter City :');

    //Designing Dynamic Query based on the inputs received from the user
    var sql = "INSERT INTO student (stud_roll, stud_name, stud_city) VALUES
    ("+ Number(roll) + ", '" + name + "', '" + city + "')";

    con.query(sql, function (err, result) {
      if (err) throw err;
      // if there is no error, you have the result
      console.log("1 record successfully inserted in the table");
    });
});
```

Output:

```
C:\Windows\System32\cmd.exe - node insert_query.js

C:\BCA>node insert_query.js
Connected!
Enter Roll Number :101
Enter Student Name :Janardan
Enter City :Pune
1 record successfully inserted in the table
```

- Assume that we have entered few more records by executing the same code. Image below shows the records currently available in the table **student**.

Table 7.1 : student

stud_roll	stud_name	stud_city
101	Janardan	Pune
102	Shivendu	Mumbai
103	Tejashri	Thane
104	Vishal	Varanasi
105	Shubhangi	Pune
106	Ashish	Nagpur

7.4 WORKING WITH SELECT COMMAND

- Now, we have some records in the table student. Let us write SELECT statement against this table to display all the available records.

Program 7.3: Program to display all the records available in the table student.

show_records.js

```
var mysql = revire('mysql');
var con = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "iccs#123",
   database: "college",
   port : '3308'
});
con.connect(function(err) {
   if (err) throw err;
   con.query("SELECT * FROM student", function (err, result, fields) {
      if (err) throw err;
      //Showing all the records
      console.log(result);
   });
});
```

Output:

```
Command Prompt - node show_records.js
C:\BCA>node show_records.js
[
  RowDataPacket {
    stud_roll: 101,
    stud_name: 'Janardan',
    stud_city: 'Pune'
  },
  RowDataPacket {
    stud_roll: 102,
    stud_name: 'Shivendu',
    stud_city: 'Mumbai'
  },
  RowDataPacket {
    stud_roll: 103,
    stud_name: 'Tejashri',
    stud_city: 'Thane'
  },
  RowDataPacket {
    stud_roll: 104,
    stud_name: 'Vishal',
    stud_city: 'Varanasi'
  },
  RowDataPacket {
    stud_roll: 105,
    stud_name: 'Shubhangi',
    stud_city: 'Pune'
  },
  RowDataPacket {
    stud_roll: 106,
    stud_name: 'Ashish',
    stud_city: 'Nagpur'
  }
```

The Result Object:

- As you can see from the output of the example above, the **result** object is an array containing each row as an object.
- To return e.g. the **name** of the third record, just refer to the third array object's as :

```
console.log("Name is", result[2].stud_name);
```

- This will show the output as **Name is Tejashri**
- We can slightly modify above code and print all the records in proper format.

Program 7.4: Program to display all the records available in the table student in table format.

show_records_format.js

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "iccs#123",
  database: "college",
  port : '3308'
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM student", function (err, result, fields) {
      if (err) throw err;
      //Showing records

      console.log("Roll No | Name | City");
      console.log("----------------------");

      //Code to iterate through an array
      Object.keys(result).forEach(function(key) {
        var record = result[key];  //Storing each row and accessing it
        console.log(record.stud_roll,"|", record.stud_name, "|",
        record.stud_city);
      });
  });
});
```

Output:

Command Prompt - node show_records_format.js

```
C:\BCA>node show_records_format.js
Roll No   | Name | City
-------------------------
101 | Janardan | Pune
102 | Shivendu | Mumbai
103 | Tejashri | Thane
104 | Vishal | Varanasi
105 | Shubhangi | Pune
106 | Ashish | Nagpur
```

7.5 UPDATING RECORDS

- Now, we have total SIX records in the table. Let us update the stud_city column. There are two records with the city name Pune. We will update value of city name from **Pune** to **Punya Nagari** using SQL update command. So, total TWO records will get updated.

Program 7.5: Program to update the student table records.

update_table_query.js

```
const prompt = require('prompt-sync')();
var mysql = require('mysql');
var con = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "iccs#123",
   database: "college",
   port : '3308'
});

con.connect(function(err) {
   if (err) throw err;
   const old_city = prompt("Enter Current City Name to be changed ::");
   const new_city = prompt("Enter New City Name ::");

   var sql = "UPDATE student SET stud_city='" + new_city + "' WHERE
   stud_city='" + old_city + "'";

   //Executing the above query
   con.query(sql, function (err, result) {
      if (err) throw err;
      console.log(result.affectedRows + " record(s) updated");
   });
});
```

Output:

```
C:\BCA>node update_table_query.js
Enter current City Name to be changed ::Pune
Enter New City Name ::Punya Nagari
2 record(s) updated
```

- **Result** object's affectedRows property returns the number of rows(records) affected due to this update query.

7.6 DELETING RECORDS

- Now, delete record(s) using SQL DELETE statement where city name is **Thane.**

Program 7.6: Program to delete the table record where city name is Thane.

delete_record.js

```
const prompt = require('prompt-sync')();

var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "iccs#123",
  database: "college",
  port: '3308'
});

con.connect(function(err) {
  if (err) throw err;

  const cityName = prompt("Enter City Name::");

  var sql = "DELETE from student WHERE stud_city ='" + cityName + "'";

  //Executing the above query
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result.affectedRows + " record(s) deleted");
  });
});
```

Output:

```
C:\BCA>node delete_record.js
Enter City Name ::Thane
1 record(s) deleted
```

Solved Programs

Program 7.7: Program to print all the records of the students who are living in the city "Pune". Now, let us assume following table **student.**

stud_roll	stud_name	stud_city	stud_per
101	Janardan	Pune	78.56
102	Shivendu	Mumbai	69.85
103	Tejashri	Thane	72.62
104	Vishal	Varanasi	65.46
105	Shubhangi	Pune	71.29
106	Ashish	Nagpur	63.98

print_city_records.js

```
const prompt = require('prompt-sync')();
var mysql = require('mysql');

var con = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "iccs#123",
   database: "college",
   port : '3308'
});

con.connect(function(err) {
   if (err) throw err;
   const cityName = prompt("Enter City Name ::");

   con.query("SELECT * FROM student where stud_city ='" + cityName + "'",
   function (err, result, fields) {
      if (err) throw err;
      //Showing records

      console.log("Roll | Name | City | Percentage");
      console.log("-------------------------------");

      //Code to iterate through an array
      Object.keys(result).forEach(function(key){
         var record = result[key]; //Storing each row and accessing it
         console.log(record.stud_roll,"|", record.stud_name, "|",
         record.stud_city, "|", record.stud_per);
      });
   });
});
```

Output:

```
C:\BCA>node print_city_records.js
Enter City Name ::Pune
Roll|     Name   | City | Percentage
--------------------------
101 | Janardan | Pune | 88.23
105 | Shubhangi| Pune | 88.69
```

Program 7.8: Program to print the merit list of the college. All students have earned marks and thus they got the percentage. Now, we will print all the records in the descending order of their percentage, which is nothing but our merit list.

Merit_List.js

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "iccs#123",
  database: "college",
  port : '3308'
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM student order by stud_per desc", function (err,
  result, fields) {
    if (err) throw err;
    //Showing records in desceding order of the percentage
    console.log("Roll | Name | City | Percentage");
    console.log("-----------------------------");

    //Code to iterate through an array
    Object.keys(result).forEach(function(key) {
        var record = result[key]; //Storing each row and accessing it
        console.log(record.stud_roll," ", record.stud_name, "|",
        record.stud_city, " ", record.stud_per);
    });
  });
});
```

Output:

```
C:\BCA>node Merit_List.js
Roll|    Name   | City     |Percentage
-----------------------------------
101 | Janardan | Pune     |  78.56
103 | Tejashri | Thane    |  72.62
105 | Shubhangi| Pune     |  71.29
102 | Shivendu | Mumbai   |  69.85
104 | Vishal   | Varanasi |  65.46
106 | Ashish   | Nagpur   |  63.98
```

Program 7.9: Program to print details of the first two toppers of the college using LIMIT statement. And also explain use of LIMIT statement.

LIMIT statement: We can limit the number of records returned from the query, by using the "LIMIT" statement:

toppers.js

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "iccs#123",
  database: "college",
  port : '3308'
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM student order by stud_per DESC LIMIT 2",
  function (err, result, fields) {
    if (err) throw err;
    //Showing TOP TWO records of the meritorious students

    console.log("Roll | Name | City | Percentage");
    console.log("-----------------------------");

    //Code to iterate through an array
    Object.keys(result).forEach(function(key) {
      var record = result[key]; //Storing each row and accessing it
      console.log(record.stud_roll,"|", record.stud_name, "|",
      record.stud_city, "|", record.stud_per);
    });
```

```
    });
});
```

Output:

```
C:\Windows\System32\cmd.exe - node toppers.js

C:\BCA>node toppers.js
Roll|    Name   | City | Percentage
-------------------------------
101 | Janardan | Pune | 78.56
103 | Tejashri | Thane| 72.62
```

Program 7.10: Program to print Players details whose name ends with string "kar". Let us consider following table **Players.**

p_id	p_name	p_city
101	Sunil Gavaskar	Mumbai
102	Kapil Dev	Chandigarth
103	Dilip Vengsarkar	Pune
104	Sachin Tendulkar	Mumbai
105	Rahul Dravid	Bengaluru

player_records.js

```
var mysql = require('mysql');

var con = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "iccs#123",
   database: "college",
   port: "3308"
});

con.connect(function(err) {
   if (err) throw err;

   con.query("SELECT * FROM players where p_name LIKE '%kar'", function
   (err, result, fields) {
      if (err) throw err;
      //Showing records of the players whose name ends with "kar"
```

```
        console.log("Player ID | Player Name | City");
        console.log("-----------------------------");

        //Code to iterate through an array
        Object.keys(result).forEach(function(key) {
            var record = result[key]; //Storing each row and accessing it
            console.log(record.p_id,"|", record.p_name, "|", record.p_city);
        });
    });
});
```

Output:

```
C:\Windows\System32\cmd.exe - node player_records.js

C:\BCA>node player_records.js
Player ID|    Player Name   | City
-------------------------------------
101      | Sunil Gavaskar   | Mumbai
103      | Dilip Vengsarkar | Pune
104      | Sachin Tendulkar | Mumbai
```

Summary

- Mostly all modern day web applications have some sort of data storage system at the backend to store data. For example, if you take the case of a web shopping application, data such as the price of an item or the number of items of a particular type would be stored in the database.
- The Node JS framework has the ability to work with databases which are commonly required by most modern day web applications.
- Node JS can work with both relational (such as Oracle and MySQL Server) and non relational databases (such as MongoDB and MySQL).

Check Your Understanding

1. _____ method() of mysql module is used to establish a connection to the database in Node JS.
   (a) makeConnection()
   (b) createConnection()
   (c) sqlConnection()
   (d) connect()
2. Method used to terminate the connection with database server is _____.
   (a) close()
   (b) exit()
   (c) end()
   (d) terminate()
3. Apart from end(), an alternative way to end the connection is to call the _____ method.

(a) destroy()
(b) exit()
(c) close()
(d) terminate()

4. Which of the following is right command to install mysql database driver in Widows OS through command prompt?
(a) npm install mssql
(b) npm install mysql
(c) npm install -mysql
(d) None of the above

5. _____ property returns the number of rows altered/updated/affected due to INSERT, DELETE or UPDATE command.
(a) affectedRows()
(b) AffectedRows
(c) AffectedRows()
(d) affectedRows

6. To execute commands against the database table, we make use of _____ method.
(a) executeQuery()
(b) query()
(c) sqlQuery()
(d) Query()

Consider this database table "Players" and answer the questions given below:

p_id	p_name	p_city	runs	wickets
101	Sunil Gavaskar	Mumbai	5682	0
102	Kapil Dev	Chandigarth	3057	434
103	Dilip Vengsarkar	Pune	8630	0
104	Sachin Tendulkar	Mumbai	15786	153
105	Rahul Dravid	Bengaluru	12695	3

7. How many records will be retrieved after executing following SQL query?
SELECT * FROM players WHERE p_name LIKE 'S%ar'
(a) 2
(b) 1
(c) 5
(d) None

8. What will be the output of following SQL query?
SELECT COUNT(DISTINCT p_city) FROM players
(a) 2
(b) 1
(c) 4
(d) 5

9. Which of the following is the best and suitable mysql query to return details of leading wicket taker?
(a) SELECT * FROM `players` ORDER BY wickets DESC LIMIT 1
(b) SELECT * FROM `players` ORDER BY wickets DESC TOP 1
(c) SELECT * FROM `players` ORDER BY wickets ASC LIMIT 1
(d) SELECT * FROM `players` where wickets = MAX(wickets)

10. How many records will be fetched by following SQL query?

```
SELECT * FROM players WHERE p_city IN ('Pune', 'Mumbai')
```

(a) 2 (b) 0

(c) 4 (d) 3

ANSWER KEY

1. (b)	2. (c)	3. (a)	4. (b)	5. (d)
6. (b)	7. (a)	8. (c)	9. (a)	10. (a)

Practice Questions

Q.I: Answer the following questions in short.

1. Explain the working of createConnection() ?
2. Explain the parameters of createConnection() ?
3. Write a sql query to fetch all details of student from student table(details: stud_roll, Stud_name, Stud_addr, stud_) ?
4. What is connection string?

Q.II: Answer the following questions.

1. Write steps to Connect with database through code?
2. Write steps to connect to database using Dashboard?
3. Write Node JS Database connectivity Program to insert record ?
4. What is Database? Explain different functions in NODE JS to connect to the MySQL database server.
5. Write a program using Node JS to create a table Players set a constraint NULL. Columns/fields should include Player_ID, Player_Name, Sports_Name, and Country_Name.
6. Write a SQL statement to create a table named jobs including columns job_id, job_title, min_salary, max_salary and check whether the max_salary amount exceeding the upper limit Rs. 15000/-
7. Write a program using Node JS to print all the distinct cities from the table customers (cust_id, cust_name, cust_mail, cust_city).
8. Write a program using Node JS to update value of city from Allahabad to Prayagraj of all the customers (cust_id, cust_name, cust_mail, cust_city).
9. Write a program using Node JS to print records of those account holders whose balance amount is between Rs. 15,000 and Rs. 25,000/-. Assume suitable table schema.

❖❖❖

OUR MOST RECOMMENDED TEXT BOOKS FOR S.Y. B.B.A. : SEMESTER-IV

• Entrepreneurship & Small Business Management	: Mrs. Arati Oturkar, Dr. Jyoti Joshi
• Production and Operations Management	: Dr. Anil Karanjkar
• Decision Making and Risk Management	: Ameya Anil Patil
• International Business Management	: Deepa Dani
• Advertising and Promotion Management	: Dr. Shaila Bootwala
• Digital Marketing	: Gautam Bapat
• Business Taxation	: Dr. Suhas Mahajan, Dr. Mahesh Kulkarni
• Financial Services	: Archana Vechalekar, Mrs. Rekha Kankariya
• HRM Functions and Practices	: Dr. Shalaka Parker, Viral Ahire
• Employment Recruitment & Record Management	: Dr. Leena Modi (Gandhi), Ankita Bhatt, Karan Rajeev Randive
• Banking and Insurance Management	: Dr. (Ms.) Girija Shankar, Vivek Datar

OUR MOST RECOMMENDED TEXT BOOKS FOR S.Y. B.B.A. (I.B.) : SEMESTER-IV

• Import Export Procedure	: Manisha Paliwal
• Research Methodology	: Ameya A. Patil, Dr. Pradnya J. Bachhav
• Business Ethics	: Sheetal Randhir & Others
• Management Information Systems	: Jayant Oke

OUR MOST RECOMMENDED TEXT BOOKS FOR S.Y. B.B.A. (C.A) : SEMESTER-IV

• Networking	: Dr. Ms. Manisha Bharambe, Vikas Tayade, Mrs. Veena Gandhi
• Object Oriented Concepts Through CPP	: Dr. Ms. Manisha Bharambe
• Operating Systems	: Gajanan A. Deshmukh, Vinayak S. More
• Advance PHP	: Swati Jadhav, Gajanan Deshmukh, Sarita Byagar
• Node JS	: Shivendu Bhushan, Pradip Waghmare

BOOKS AVAILABLE AT

PRAGATI BOOK CENTER - Email: pbcpune@pragationline.com

- 157 Budhwar Peth, Opp. Ratan Talkies, Next To Balaji Mandir, **Pune** 411002 • **Mobile** : 9657703148
- 676/B Budhwar Peth, Opp. Jogeshwari Mandir, **Pune** 411002 Tel : (020) 2448 7459 • **Mobile** : 9657703147 / 9657703149
- 152 Budhwar Peth, Near Jogeshwari Mandir, **Pune** 411002 **Mobile** : 8087881795

PRAGATI BOOK CORNER - Email: niralimumbai@pragationline.com

- Apurva Building, Shop No. 1, Bhavani Shankar Road, Opp. Shardashram Society, Dadar (W), Mumbai 400028. Tel: (022) 2422 3526/6662 5254 • Mobile : 9819935759

niralipune@pragationline.com | www.pragationline.com

Also find us on www.facebook.com/niralibooks @nirali.prakashan

N4957