Roll No.: **58/A** Exam Seat No.:_____

Vivekanand Education Society's
Institute of Technology

(An Autonomous Institute Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)
Hashu Advani Memorial Complex, Collector's Colony, R. C.Marg, Chembur, Mumbai – 400074.

VESIT V.E.S. Since 1962

CERTIFICATE

Certified that Mr. **Om Bhagwandas Wadhwani** .
of **FYMCA - A** has satisfactorily completed a course of the necessary experiments in **Web Application Technologies Lab** ,
under my supervision in the Institute of Technology in the academic year **2025-2026**.

Principal

Head of Department
(Dr. Shivkumar Goel)

Faculty Incharge
(Mr. Sunny Nahar)

External Examiner

V.E.S. Institute of Technology
Collector Colony, Chembur, Mumbai, Maharashtra 400074
Department of M.C.A

INDEX

Sr. No	Contents	Date of Preparation	Date of Submission	Marks	Sign
01.	Introduction to Node.js, Advantages and Disadvantages, Node.js Process Model, Traditional Web Server Model, Installation.	03/09/2025	09/09/2025		
02.	Create an application to demonstrate Node.js Modules	09/09/2025	16/09/2025		
03.	Create an application to demonstrate various Node.js Events in the event emitter class.	16/09/2025	23/09/2025		
04.	Create an application to demonstrate Node.js Functions-timer function (displays every 10 seconds)	23/09/2025	30/09/2025		
05.	Using File Handling demonstrate all basic file operations (Create, write, read, delete)	30/09/2025	14/10/2025		
06.	Implementing a Node.js HTTP Server for File Serving, Video Streaming & Routing Multiple Webpages	14/10/2025	28/10/2025		
07.	To develop a Node.js application that connects to a MySQL database and performs basic CRUD operations.	28/10/2025	11/11/2025		
08.	To install and set up a TypeScript environment and write programs using decisions, functions, and classes.	28/10/2025	11/11/2025		
09.	To study Angular basics, set up its development environment, understand its architecture, and create an application demonstrating directives and pipes.	11/11/2025	18/11/2025		
10.	Demonstrate features of Angular forms with a program	18/11/2025	26/11/2025		
11.	Create an application to demonstrate SPA	26/11/2025	03/12/2025		
12.	Assignment 01				
13.	Assignment 02				

Final Grade	Instructor Signature

Name of Students : Om Bhagwandas Wadhwani			
Roll Number : 58/A		**Lab Assignment Number : 1**	
Title of Lab Assignment : Introduction to Node.js			
DOP:03/09/2025		**DOS: 09/09/2025**	
CO Mapped: **CO1**	**PO Mapped:** **PO3, PO4,** **PSO1, PSO2**	**Faculty** **Signature:**	**Marks:**

Aim : Introduction to Node.js , Advantages and Disadvantages, Node.js Process Model, Traditional Web Server Model, Installation

Description :

Introduction to Node.js:

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to run JavaScript code outside of the web browser.
It was created by Ryan Dahl in 2009 and is built on Google Chrome's V8 engine, which compiles JavaScript into machine code for fast execution.
Node.js is particularly useful for building:
Real-time applications (e.g., chat apps, live updates).
APIs and microservices.
Streaming applications.
Its design focuses on asynchronous, event-driven programming which makes it efficient for I/O-heavy operations (like file systems, databases, and APIs).

Advantages of Node.js:

Asynchronous & Non-blocking – Handles multiple requests simultaneously.
Fast execution – Powered by the V8 engine.
Single programming language – JavaScript for both frontend & backend.
Scalable – Suitable for microservices and real-time applications.
Huge community & npm support – Rich ecosystem of packages.

Disadvantages of Node.js:

Not ideal for CPU-intensive tasks (e.g., image processing, ML computations).
Callback hell (mitigated by Promises/async-await).
Rapidly changing environments – frequent updates can cause compatibility issues.
Single-threaded – heavy computations can block event loop.

REPL:

REPL stands for Read-Eval-Print Loop. It is an interactive shell that processes Node.js expressions, debugs in real-time and outputs the result.
To access the REPL, simply type node in the cmd after Node.js is installed.

```
PS C:\Users\user> node
Welcome to Node.js v22.12.0.
Type ".help" for more information.
>
```

Execute any JS command.
For example, console.log("REPL Example");

console : The console object in Node.js provides methods like log(), error(), and warn() for outputting information and debugging. It is a global object, available by default, and is commonly used to log messages and inspect values during program execution.

log() :A method of console which outputs the message passed in the parentheses. The message may be a String ,variable value, objects, etc.

```
PS C:\Users\user> node
Welcome to Node.js v22.12.0.
Type ".help" for more information.
> console.log("REPL Example")
REPL Example
undefined
>
```

Node.js Process Model:

Single-threaded with event loop.
Uses asynchronous callbacks to handle requests.
Employs non-blocking I/O so one thread can serve thousands of requests.
Offloads CPU-intensive tasks to worker threads / child processes when needed.

Traditional Web Server Model vs Node.js

Traditional Model:

Multi-threaded.
Each request creates a **new thread/process**.
High memory & CPU usage with many clients.

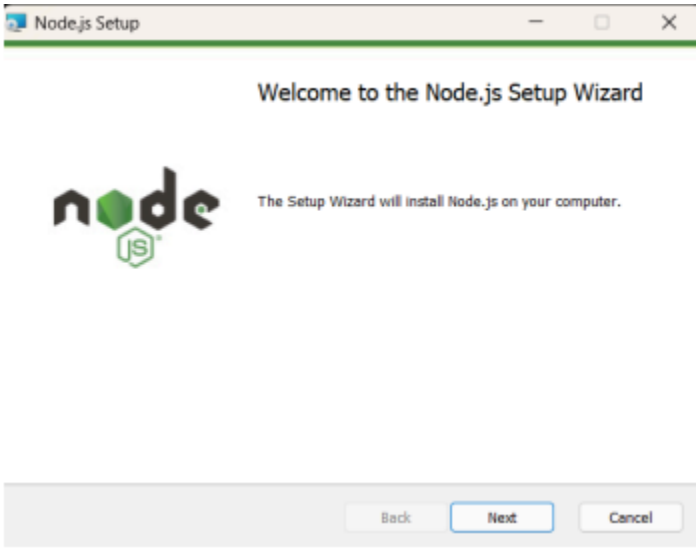
Node.js Model:

Single-threaded event loop.
Handles multiple requests with **non-blocking I/O**.
Lightweight and efficient for I/O-heavy apps.

Installation:

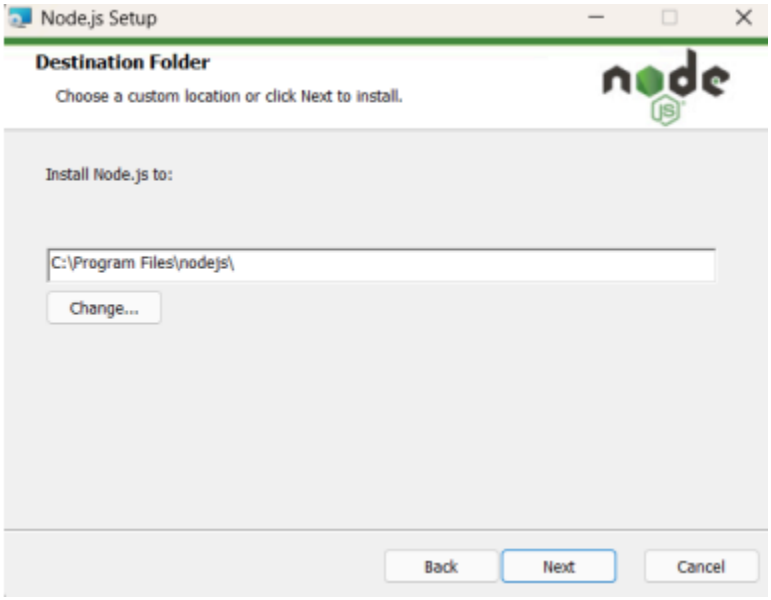
Step 1: Download the latest version of Node.js installer(.msi) from Node.js website. Select the version(Latest 22.19.0) you want to install.

Step 2: Double click on the .msi installer. A Node.js Setup Wizard will pop up. Click on Next.



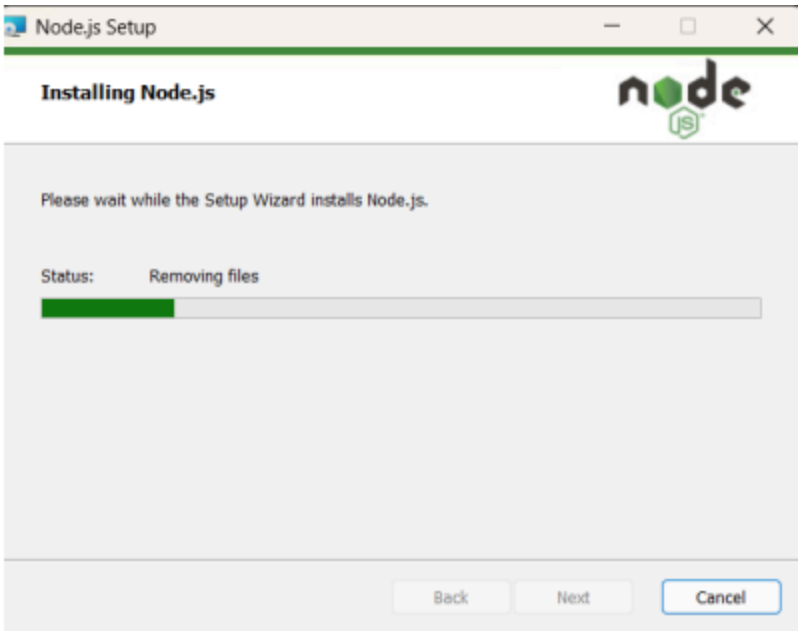
Step3: After clicking on Next, the End–User License Agreement will open. Click on “I accept” and then next.

Step 4: Further, Choose a custom location or click Next to install.



Step 5: Next, a Custom Setup window appears. Click on Next.

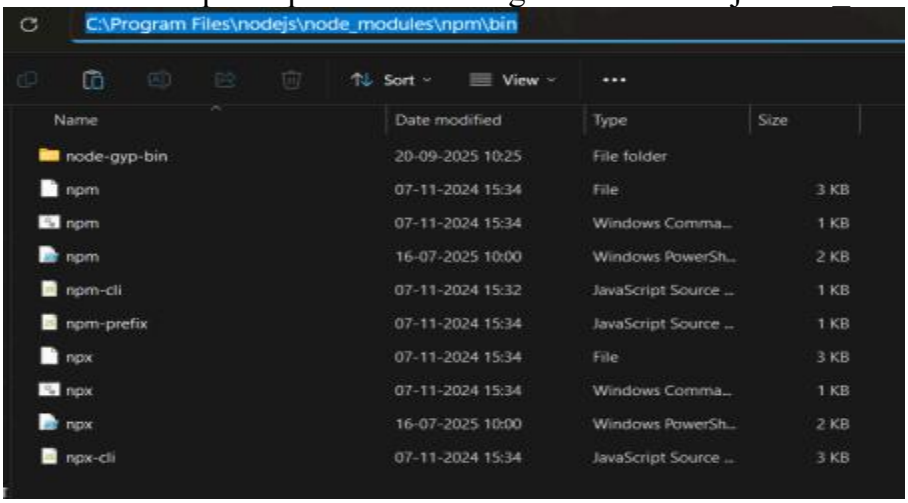
Step 6: In addition to this, a prompt stating “Ready to install Node.js” appears. Click on Install.



Step 7: Next, the window prompt shows that you have Completed the Node.js Setup Wizard. Click on Finish.

Step 8:

Select the file path up to Bin: C:\Program Files\nodejs\node_modules\npm\bin



Step 9: Set the environment variables so that Node.js is available globally inside the system. Click on This PC -> Properties -> Settings Tab appears -> Go to Search Option and search “environment variables” -> Select “Edit the system environment variables”

Step 10: Now click on “Environment Variables” -> Select Path under “User variables for system” -> Click on Edit -> Click on New -> Enter the file path “C:\Program Files\nodejs\node_modules\npm\bin” -> OK -> OK -> OK

Step 11: Node.js setup is completed. To verify that Node.js is available for global use, Opencmd and type node - - version. You should be able to see your downloaded version

```
C:\Users\user>node --version
v20.16.0
```

Program 1: Print Today’s date and time using REPL.

Code:

```
const now = new Date()
console.log("Date and Time", now);
console.log("Year:", now.getFullYear());
console.log("Month:", now.getMonth());
console.log("Date:", now.getDate());
console.log("Hours:", now.getHours());
console.log("Minutes:", now.getMinutes());
console.log("Seconds:", now.getSeconds());
```

Output:

```
PS C:\Users\Student.MCAB14-21\Desktop\node.js> node .\question1.js
Date and Time 2025-09-18T06:29:25.134Z
Year: 2025
Month: 8
Date: 18
Hours: 11
Minutes: 59
Seconds: 25
PS C:\Users\Student.MCAB14-21\Desktop\node.js>
```

Program 2 : Write a program to print the given pattern.

```
1 2 3 4 5
 1 2 3 4
 1 2 3
 1 2
 1
```

Code:

```
function printPattern(){
    for(let i=5;i>=1;i--){
    let out="";
        for (let j = 1;j<=i;j++){
            out+=j;
        }
            console.log(out);
            out="";
    }
}
printPattern();
```

Output:

```
PS C:\Users\Student.MCAB14-21\Desktop\node.js> node .\pattern.js
12345
1234
123
12
1
```

Program 3 : Write a program to print the first 20 Fibonacci numbers (take input through the command line).

Code:

```
function isPrime(num) {
    if (num <= 1) return false;
    for (let i = 2; i <= Math.sqrt(num); i++) {
        if (num % i === 0) {
            return false;
        }
    }
    return true;
}
for (let i = 1; i <= 100; i++) {
    if (isPrime(i)) {
        console.log(i);
    }
}
```

Output:

```
PS C:\Users\Student.MCAB14-21\Desktop\node.js> node .\prime.js
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
```

Program 4 : Write a program to prime numbers between 1 to 100.

Code:

```
const readline =require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
  });

  readline.question('Enter Number ', n => {
    let a = 0, b = 1;
```

```
    console.log(a);
    while (b <= +n) {
      console.log(b);
      [a, b] = [b, a + b];
    }
    readline.close();
});
```

Output:

```
PS C:\Users\Student.MCAB14-21\Desktop\node.js> node .\fib.js
Enter Number 20
0
1
1
2
3
5
8
13
```

Conclusion :

The above programs demonstrate the fundamental usage of Node.js for handling basic operations using loops, conditional statements, and the REPL environment. Leveraging Chrome's V8 engine, Node.js ensures code readability, simplicity, and efficiency, forming a solid foundation for building advanced applications.

Name of Students : Om Bhagwandas Wadhwani			
Roll Number : 58/A		**Lab Assignment Number: 2**	
Title of Lab Assignment : Create an application to demonstrate Node.js Modules Built in Module			
DOP : 09/09/2025		**DOS : 16/09/2025**	
CO Mapped : CO2	**PO Mapped : PO5, PSO1**	**Faculty Signature :**	**Marks :**

AIM:Create an application to demonstrate Node.js Modules Built in Module

Description:

Modules in Node.js:

Modules are the building blocks of Node.js applications, allowing you to organize code into logical, reusable components. They help in:
Organizing code into manageable files
Encapsulating functionality
Preventing global namespace pollution
Improving code maintainability and reusability

Types of Modules:

Core / Built-in Module:

In Node.js, **core modules** (also called **built-in modules**) are the modules that come pre-installed with the Node.js runtime. They provide essential functionalities to perform a wide range of operations without the need to install any additional packages. These modules can be directly imported using the `require()` function.

Custom / Local Module:

A **custom module** in Node.js is a user-defined JavaScript file that contains reusable code such as functions, variables, or classes. Unlike built-in modules, custom modules are created by developers to organize and reuse logic across multiple files. They are loaded into an application using the `require()` function and exported from a file using `module.exports` or `exports`.

Third Party Module:

Third-party modules are Node.js modules that are developed and published by the community or external developers. They are not part of the core Node.js runtime and need to be installed via the **Node Package Manager (npm)** before use. These modules provide additional functionalities that simplify development, such as working with databases, handling HTTP requests, or adding authentication.

Advantages: Node.js modules promote code reusability, modularity, maintainability, and faster development through built-in and third-party modules.

Disadvantages: Excessive dependencies, version conflicts, performance overhead, and potential security risks are key drawbacks.

Program 1 : Write a program to print information about the computer's operating system using OS module. (Use any 5 module)

Code:

```
const os = require('os');
console.log('OS Type:', os.type());
console.log('Platform:', os.platform());
console.log('CPU Info:', os.cpus()[0].model);
console.log('Home Directory:', os.homedir());
console.log('Uptime in seconds:', os.uptime());
```

Output:

```
PS C:\Users\Student\Desktop> node test.js
OS Type: Windows_NT
Platform: win32
CPU Info: 12th Gen Intel(R) Core(TM) i5-12400
Home Directory: C:\Users\Student
Uptime in seconds: 254064.812
PS C:\Users\Student\Desktop>
```

Program 2 : Print "Hello" every 500 milliseconds using the Timer Module. The message should be printed exactly 10 times. Use setInterval, ClearInterval and setTimeout methods.

Code:

```
let count = 0;
const interval = setInterval(() => {
  console.log('Hello');
  count++;
  if (count === 10) {
    clearInterval(interval);
  }
}, 500);
```

Output:

```
PS C:\Users\Student\Desktop> node test.js
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
PS C:\Users\Student\Desktop>
```

Program 3 : Create a Calculator Node.js module with functions add, subtract, multiply and divide. And use the Calculator module in another Node.js file.

Code:

Calculator.js

```
function add(a, b) {
    return a + b;
}
function subtract(a, b) {
    return a - b;
}
function multiply(a, b) {
    return a * b;
}
function divide(a, b) {
    if (b === 0) {
        throw new Error("Division by zero is not allowed.");
    }
    return a / b;
}
module.exports = { add, subtract, multiply, divide };
```

app.js

```
const readline = require('readline');
const calculator = require('./calculator');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});
```

```
rl.question("Enter first number: ", (num1) => {
  rl.question("Enter second number: ", (num2) => {
    rl.question("Choose operation (add, subtract, multiply, divide): ", (operation) => {
      const a = parseFloat(num1);
      const b = parseFloat(num2);
      let result;
      try {
        switch (operation.toLowerCase()) {
          case 'add':
            result = calculator.add(a, b);
            break;
          case 'subtract':
            result = calculator.subtract(a, b);
            break;
          case 'multiply':
            result = calculator.multiply(a, b);
            break;
          case 'divide':
            result = calculator.divide(a, b);
            break;
          default:
            console.log("Invalid operation!");
            rl.close();
            return;
        }

        console.log(`Result: ${result}`);
      } catch (error) {
        console.error(error.message);
      }

      rl.close();
    });
  });
});
```

Output:

```
PS D:\WebPracticals\Web> node app.js
Enter first number: 2
Enter second number: 3
Choose operation (add, subtract, multiply, divide): add
Result: 5
```

Program 4 : Create a circle module with functions to find the area and perimeter of a circle and use it.

Code:

circle.js

```
const PI = Math.PI;
function area(radius) {
   return PI * radius * radius;
}
function perimeter(radius) {
   return 2 * PI * radius;
}
module.exports = { area, perimeter };
```

app1.js

```
const readline = require('readline');
const circle = require('./circle');
const rl = readline.createInterface({
   input: process.stdin,
   output: process.stdout
});
rl.question("Enter the radius of the circle: ", (r) => {
   const radius = parseFloat(r);

   if (isNaN(radius) || radius <= 0) {
      console.log("Please enter a valid positive number for radius.");
   } else {
      console.log(`Area of circle: ${circle.area(radius).toFixed(2)}`);
      console.log(`Perimeter of circle: ${circle.perimeter(radius).toFixed(2)}`);
   }

   rl.close();
});
```

Output:

```
PS D:\WebPracticals\Web> node .\app1.js
Enter the radius of the circle: 7
Area of circle: 153.94
Perimeter of circle: 43.98
```

Conclusion :

Modules are a core feature of Node.js that enable developers to break down programs into smaller, reusable components. Built-in modules such as os, fs, and http provide direct access to system-level functionalities, while custom modules allow developers to create and organize their own reusable code. This modular approach enhances efficiency, code reusability, and maintainability, thereby making Node.js a powerful platform for scalable application development.

<table>
<tr><td colspan="4">Name of Students : Om Bhagwandas Wadhwani</td></tr>
<tr><td colspan="2">Roll Number : 58/A</td><td colspan="2">Lab Assignment Number:3</td></tr>
<tr><td colspan="4">Title of Lab Assignment :Demonstration of Node.js Events using Event Emitter Class</td></tr>
<tr><td colspan="2">DOP:16/09/2025</td><td colspan="2">DOS: 23/09/2025</td></tr>
<tr><td>CO Mapped:
CO1</td><td>PO Mapped:
PO3, PO4,
PSO1, PSO2</td><td>Faculty
Signature:</td><td>Marks:</td></tr>
</table>

Aim: To create a Node.js application that demonstrates the use of the EventEmitter class by defining and triggering custom events for sorting, reversing, and searching elements in an array.

Description::

In Node.js, events are actions or occurrences that can be detected and handled asynchronously.

- The EventEmitter class (part of the built-in events module) allows us to create, register, and trigger custom events.
- We can associate a function (event listener) with an event and invoke it using the emit() method.

Key methods used:

- `on(eventName, listener)` → `Registers an event listener.`
- `emit(eventName, args)` → `Triggers the event.`
- `removeListener(eventName, listener)` → `Removes an event listener.`

In this experiment, we use events to perform three operations on an array:

1. Sort elements
2. Reverse elements
3. Search for a specific element

Program:

1. Create an application to demonstrate various Node.js Events in Event emitter class. Create functions to sort, reverse and search for an element in an array. Register and trigger these functions using events.

Description:

In this program, the EventEmitter class from Node.js's built-in events module is used to create and handle custom events. An object named emitter is created to manage the events. A sample array of numbers is defined, and three separate functions—sortArray(), reverseArray(), and searchElement()—are written to perform sorting, reversing, and searching operations respectively. Each function is then registered as an event listener

using the emitter.on() method with unique event names (sortEvent, reverseEvent, searchEvent). Finally, these events are triggered using the emitter.emit() method, which executes the corresponding functions and displays the results for each operation on the array. This demonstrates how Node.js uses event-driven programming to execute specific tasks efficiently.

Code:

```
const EventEmitter = require('events');
const emitter = new EventEmitter();
let arr = [23, 5, 78, 12, 56, 9];
function sortArray() {
console.log("Original Array:", arr);
let sorted = [...arr].sort((a, b) => a - b);
console.log("Sorted Array:", sorted);
}
function reverseArray() {
let reversed = [...arr].reverse();
console.log("Reversed Array:", reversed);
}
function searchElement(element) {
let found = arr.includes(element);
console.log(found ? `Element ${element} found in array.` : `Element ${element} not
found.`);
}
// Registering events
emitter.on('sortEvent', sortArray);
emitter.on('reverseEvent', reverseArray);
emitter.on('searchEvent', searchElement);
// Triggering events
emitter.emit('sortEvent');
emitter.emit('reverseEvent');
emitter.emit('searchEvent', 12);
```

Output:

```
Microsoft Windows [Version 10.0.19045.6332]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Student>Desktop
'Desktop' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Student>cd Desktop

C:\Users\Student\Desktop>node PRAC3
Original Array: [ 23, 5, 78, 12, 56, 9 ]
Sorted Array: [ 5, 9, 12, 23, 56, 78 ]
Reversed Array: [ 9, 56, 12, 78, 5, 23 ]
Element 12 found in array.

C:\Users\Student\Desktop>
```

Conclusion:

In this practical, we learned how to use the EventEmitter class in Node.js to handle and trigger custom events. By registering functions for sorting, reversing, and searching elements in an array, we understood the event-driven architecture of Node.js. This experiment demonstrates how Node.js efficiently executes asynchronous event-based programming.

<table>
<tr><td colspan="4">Name of Student : Om Bhagwandas Wadhwani</td></tr>
<tr><td colspan="2">Roll Number : 58/A</td><td colspan="2">LAB Assignment Number:04</td></tr>
<tr><td colspan="4">Title of LAB Assignment : Create an application to demonstrate Node.js Functions-timer function</td></tr>
<tr><td colspan="2">DOP : 23/09/2025</td><td colspan="2">DOS :30/09/2025</td></tr>
<tr><td>CO Mapped : CO1</td><td>PO Mapped: PO3, PO4 ,PSO1 ,PSO 2</td><td>Faculty Signature:</td><td>Marks :</td></tr>
</table>

AIM: Create an application to demonstrate Node.js Functions-timer function(displays every 10 seconds)

DESCRIPTION:

Node.js Timer Functions:

The Node.js Timers module provides a global API for scheduling functions (callbacks) to be executed at a future time. These functions are part of Node.js's asynchronous programming model, enabling non-blocking operations. They are global, meaning you do not need to use require() to access them.

- **Synchronous:**

Code execution is blocking. The program must wait for a synchronous function to complete its task and return a result before moving on to the next line of code.Wait and Block: Tasks run in sequence.

- **Asynchronous:**

Code execution is non-blocking. The function is initiated, and the program moves immediately to the next line. The function's result is handled later using a callback, a Promise, or Async/Await syntax once the task (like I/O or a timer) is finished.Start and Continue: Tasks run concurrently with other code.

The core timer functions can be categorized into Scheduling Timers and Canceling Timers.

- **setTimeout(callback, delay):**
  - To execute a function once after a minimum specified delay in milliseconds.
  - Executes in the Timers phase of the Event Loop after the delay has passed.
- **setInterval(callback, delay):**
  - To execute a function repeatedly at a minimum interval of delay milliseconds.
  - Executes in the Timers phase of the Event Loop every time the interval is reached
- **setImmediate(callback):**
  - To execute a function asynchronously, but as soon as possible after the current I/O phase has completed (i.e., in the very next "check" phase of the Event Loop)
  - Executes in the Check phase of the Event Loop, typically before any

pending setTimeout callbacks set to delay=0.

CODE:

```
function displayMessage() {

 const currentTime = new Date().toLocaleTimeString();
 console.log(`[${currentTime}] - This message is displayed every 10 seconds.`);
}
const intervalId = setInterval(displayMessage, 10000);
console.log("Node.js Timer Application Started.");
console.log("The message will display every 10 seconds. Press Ctrl+C to stop the
application.");
```

output:

```
C:\>D:

D:\>node filesave.js
Node.js Timer Application Started.
The message will display every 10 seconds. Press Ctrl+C to stop the application.
[1:17:56 pm] - This message is displayed every 10 seconds.
[1:18:06 pm] - This message is displayed every 10 seconds.
[1:18:16 pm] - This message is displayed every 10 seconds.
[1:18:26 pm] - This message is displayed every 10 seconds.
[1:18:36 pm] - This message is displayed every 10 seconds.
^C
D:\>
```

Conclusion:

Node.js Timers (setTimeout, setInterval, setImmediate) are the asynchronous scheduling core. They defer callback execution to keep the Event Loop non-blocking, ensuring application responsiveness.

<table>
<tr><td colspan="4">Name of Student : Om Bhagwandas Wadhwani</td></tr>
<tr><td colspan="2">Roll Number : 58/A</td><td colspan="2">LAB Assignment Number : 05</td></tr>
<tr><td colspan="4">Title of LAB Assignment : Using File Handling demonstrate all basic file operations (Create, write, read, delete)</td></tr>
<tr><td colspan="2">DOP : 30/09/2025</td><td colspan="2">DOS :14/10/2025</td></tr>
<tr><td>CO Mapped :
CO1</td><td>PO Mapped:
PO3 , PO4, PSO1,PSO2</td><td>Faculty Signature:</td><td>Marks :</td></tr>
</table>

AIM: Using File Handling demonstrate all basic file operations (Create, write, read, delete)

Description:

Node.js's File System (fs) module enables communication with the computer's file system. File creation, writing, reading, appending, renaming, and deletion are all possible with this module.
This software shows how to:
- Create and write data into a file
- Read data from the file
- After reading the file, delete it

All of these activities are carried out asynchronously, with the results handled via callback functions when each operation is finished.

Problem Statement : Using File Handling demonstrate all basic file operations (Create, write, read, delete)

CODE: CREATE,WRITE

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

try {
  fs.writeFileSync('example.txt', 'Hello, Node.js!');
  console.log('File created and data written!');
} catch (err) {
  console.error('Error writing file:', err);
}
```

OUTPUT:

```
Command Prompt
Microsoft Windows [Version 10.0.19045.6216]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>cd Dsektop
The system cannot find the path specified.

C:\Users\user>cd Desktop

C:\Users\user\Desktop>node create.js
File created and data written!

C:\Users\user\Desktop>
```

CODE: READ

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

OUTPUT:

```
C:\Users\user\Desktop> node read.js
File content: Hello, Node.js!
```

CODE:DELETE

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

fs.unlink('example.txt', (err) => {
  if (err) {
    console.error('Error deleting file:', err);
    return;
  }
  console.log('File deleted successfully!');
});
```

OUTPUTt:

```
C:\Users\user\Desktop>
C:\Users\user\Desktop>node delete.js
File deleted successfully!

C:\Users\user\Desktop>
```

Conclusion:

Using the fs module, this program effectively illustrates fundamental file handling functions in Node.js. Using callback functions, it effectively creates, reads, and deletes files asynchronously.

Name of the student: Om Bhagwandas Wadhwani			
Roll No. 58/A		**Lab Practical No. 06**	
Title of Lab Practical: i) To create HTTP Server in node.js that serve different types of files,stream video using piping and render multipleHTML pages using routing			
DOP: 14/10/2025		**DOS: 28/10/2025**	
CO Mapped: CO2	**PO Mapped: PO1,PO3, PO4**	**Faculty Signature:**	**Marks:**

AIM : i) Create a HTTP Server and serve HTML,CSV,JSON and PDF Files.
ii) Create a HTTP Server and stream a video file using piping.
iii) Develop 3 HTML Web Pages for college home page(write about college & dept),about me, contact.Create a server and render these pages using Routing.

Description:

1.Create a HTTP Server and serve HTML,CSV,JSON and PDF Files:

An HTTP server is software that listens for client requests (usually from web browsers) and responds using the HTTP (HyperText Transfer Protocol).
It delivers different types of files such as:

- HTML webpages
- CSV (Comma Separated Values) data files
- JSON API data
- PDF documents

When a user enters a URL in the browser, the HTTP server receives the request and sends the requested file along with a correct Content-Type header.

2.Create a HTTP Server and stream a video file using piping.

Creating an HTTP server to stream a video file using piping means setting up a server that reads the video in small chunks and sends it to the client continuously instead of loading the entire file at once. This allows smooth playback, reduces memory usage, and supports real-time streaming as the video is being transferred through a data stream (pipe).

3.Develop 3 HTML Web Pages for college home page(write about college & dept),about me, contact.Create a server and render these pages using Routing.

This task involves creating three HTML pages: a college home page (with details about the college and department), an “About Me” page, and a “Contact” page. A web server is then created with routing so that each URL loads the correct page—for example, / shows the home page, /about shows the About Me page, and /contact shows the contact page. Routing helps the server render the right HTML file based on the user’s request.

Code:

i) Create a HTTP Server and serve HTML,CSV,JSON and PDF Files.

server1.js

```
const http = require("http");
const fs = require("fs");
const path = require("path");

const PORT = 3000;

http.createServer((req, res) => {
  console.log("Request for:", req.url);

  if (req.url === "/" || req.url === "/index.html") {
    fs.readFile("index.html", (err, data) => {
      if (err) {
        res.writeHead(404);
        res.end("HTML file not found");
      } else {
        res.writeHead(200, { "Content-Type": "text/html" });
        res.end(data);
      }
    });

  } else if (req.url === "/data.csv") {
    fs.readFile("data.csv", (err, data) => {
      if (err) {
        res.writeHead(404);
        res.end("CSV file not found");
      } else {
        res.writeHead(200, { "Content-Type": "text/csv" });
        res.end(data);
      }
    });

  } else if (req.url === "/data.json") {
    fs.readFile("data.json", (err, data) => {
      if (err) {
```

```
      res.writeHead(404);
      res.end("JSON file not found");
     } else {
      res.writeHead(200, { "Content-Type": "application/json" });
      res.end(data);
     }
   });

 } else if (req.url === "/document.pdf") {
   fs.readFile("document.pdf", (err, data) => {
    if (err) {
      res.writeHead(404);
      res.end("PDF file not found");
     } else {
      res.writeHead(200, { "Content-Type": "application/pdf" });
      res.end(data);
     }
   });

 } else {
   res.writeHead(404, { "Content-Type": "text/plain" });
   res.end("404 Not Found");
 }
}).listen(PORT, () => {
 console.log(`Server running at http://localhost:${PORT}`);
});
```

index.html

```
<!DOCTYPE html>
<html>
<head>
 <title>Node.js File Server</title>
</head>
<body>
 <h1>Welcome to Node.js File Server</h1>
 <ul>
  <li><a href="/data.csv">Download CSV</a></li>
```

```
    <li><a href="/data.json">View JSON</a></li>
    <li><a href="/document.pdf">Open PDF</a></li>
  </ul>
</body>
</html>
```

data.csv

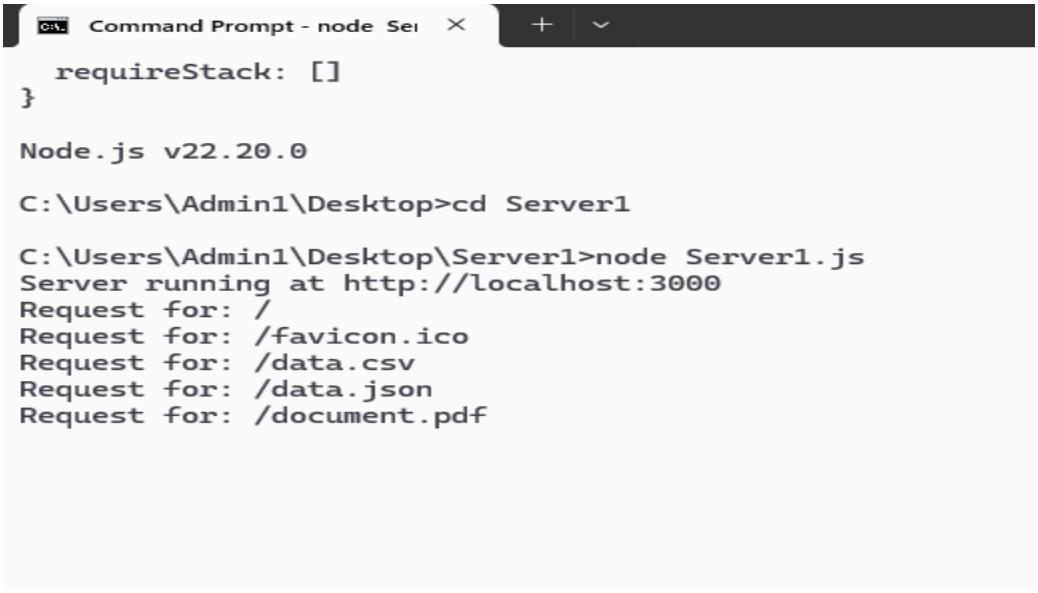
```
name,age,city
Alice,30,New York
Bob,25,Los Angeles
Charlie,35,Chicago
```

data.json

```
name,age,city
Alice,30,New York
Bob,25,Los Angeles
Charlie,35,Chicago
```

Untitled.pdf

Output:



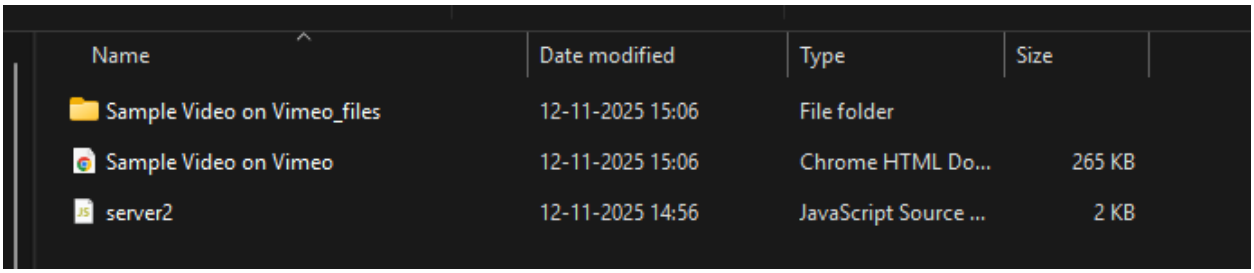
ii) Create a HTTP Server and stream a video file using piping.

code:

```
// Simple Node.js HTTP server to stream a video using piping

const http = require("http");
const fs = require("fs");
const path = require("path");

const PORT = 3000;

http.createServer((req, res) => {
  if (req.url === "/video") {
    const videoPath = path.join(__dirname, "video.mp4");
    const stat = fs.statSync(videoPath);
    const fileSize = stat.size;
    const range = req.headers.range;

    if (range) {
      // Parse the Range header for partial content
      const parts = range.replace(/bytes=/, "").split("-");
      const start = parseInt(parts[0], 10);
      const end = parts[1] ? parseInt(parts[1], 10) : fileSize - 1;

      const chunksize = end - start + 1;
```

```
    const file = fs.createReadStream(videoPath, { start, end });
    const head = {
      "Content-Range": `bytes ${start}-${end}/${fileSize}`,
      "Accept-Ranges": "bytes",
      "Content-Length": chunksize,
      "Content-Type": "video/mp4",
    };

    res.writeHead(206, head);
    file.pipe(res);
  } else {
    // Send the whole video if no range is requested
    const head = {
      "Content-Length": fileSize,
      "Content-Type": "video/mp4",
    };
    res.writeHead(200, head);
    fs.createReadStream(videoPath).pipe(res);
  }
 } else {
  // Basic home page
  res.writeHead(200, { "Content-Type": "text/html" });
  res.end(`
    <h1>Video Streaming Server</h1>
    <video width="640" height="360" controls>
      <source src="/video" type="video/mp4">
      Your browser does not support the video tag.
    </video>
  `);
 }
}).listen(PORT, () => {
 console.log(`Server running at http://localhost:${PORT}`);
});
```

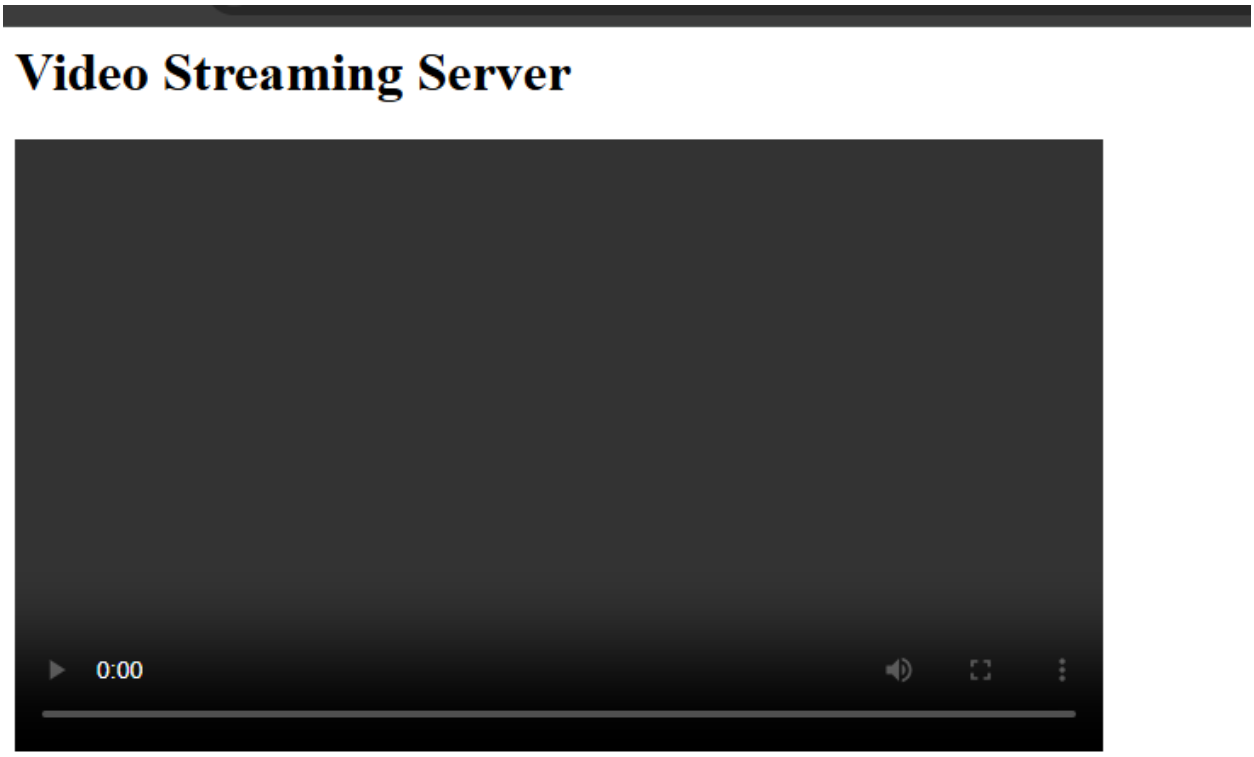
Download video for sample



Name	Date modified	Type	Size
Sample Video on Vimeo_files	12-11-2025 15:06	File folder	
Sample Video on Vimeo	12-11-2025 15:06	Chrome HTML Do...	265 KB
server2	12-11-2025 14:56	JavaScript Source ...	2 KB

Output:

```
C:\Users\Admin1>cd Desktop

C:\Users\Admin1\Desktop>cd server2

C:\Users\Admin1\Desktop\server2>node server2.js
Server running at http://localhost:3000
node:fs:1739
  const stats = binding.stat(
                        ^

Error: ENOENT: no such file or directory, stat 'C:\Users\Admin1\Desktop\server2\video.mp4'
    at Object.statSync (node:fs:1739:25)
    at Server.<anonymous> (C:\Users\Admin1\Desktop\server2\server2.js:12:21)
    at Server.emit (node:events:519:28)
    at parserOnIncoming (node:_http_server:1173:12)
    at HTTPParser.parserOnHeadersComplete (node:_http_common:121:17) {
  errno: -4058,
  code: 'ENOENT',
  syscall: 'stat',
  path: 'C:\\Users\\Admin1\\Desktop\\server2\\video.mp4'
}

Node.js v22.20.0

C:\Users\Admin1\Desktop\server2>https://sample-videos.com/video321/mp4/720/sample-5s.mp4
```



iii) Develop 3 HTML Web Pages for college home page(write about college & dept),about me, contact.Create a server and render these pages using Routing.
Code:
server3.js

```
// Node.js HTTP Server with Routing (3 Pages)

const http = require("http");
const fs = require("fs");
const path = require("path");

const PORT = 3000;

// Function to serve a file
function serveFile(filePath, contentType, res) {
  fs.readFile(filePath, (err, data) => {
    if (err) {
      res.writeHead(404, { "Content-Type": "text/plain" });
      res.end("404 - Page Not Found");
```

```
    } else {
      res.writeHead(200, { "Content-Type": contentType });
      res.end(data);
    }
  });
}

// Create server with routing
http.createServer((req, res) => {
  console.log(`Request for ${req.url}`);

  if (req.url === "/" || req.url === "/home") {
    serveFile(path.join(__dirname, "index.html"), "text/html", res);
  } else if (req.url === "/about") {
    serveFile(path.join(__dirname, "about.html"), "text/html", res);
  } else if (req.url === "/contact") {
    serveFile(path.join(__dirname, "contact.html"), "text/html", res);
  } else {
    res.writeHead(404, { "Content-Type": "text/html" });
    res.end("<h1>404 Page Not Found</h1>");
  }
}).listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>My College - Home</title>
</head>
<body style="font-family:Arial;">
  <h1>Welcome to My College</h1>
  <p>Our college is a premier institution offering high-quality education across various
departments.</p>
  <h2>Departments</h2>
  <ul>
    <li>Computer Science</li>
```

```
    <li>Electronics</li>
    <li>Mechanical Engineering</li>
    <li>Civil Engineering</li>
  </ul>
  <nav>
    <a href="/about">About Me</a> |
    <a href="/contact">Contact</a>
  </nav>
</body>
</html>
```

about.html

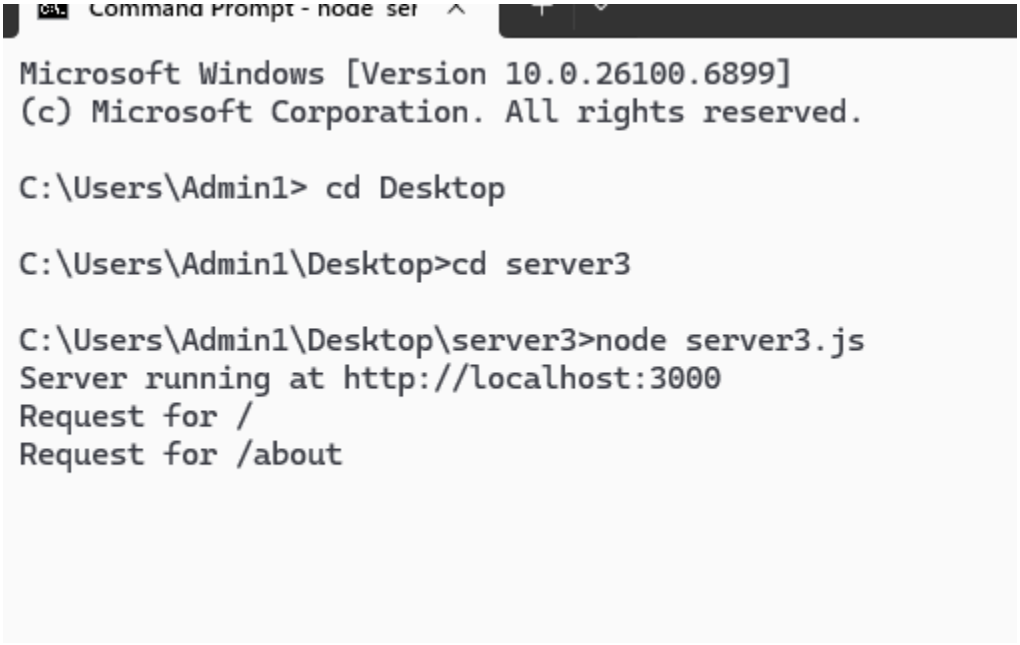
```
<!DOCTYPE html>
<html>
<head>
  <title>About Me</title>
</head>
<body style="font-family:Arial;">
  <h1>About Me</h1>
  <p>Hello! My name is <b>John Doe</b>, a student at My College.</p>
  <p>I am passionate about web development and enjoy learning Node.js and backend
technologies.</p>
  <nav>
    <a href="/">Home</a> |
    <a href="/contact">Contact</a>
  </nav>
</body>
</html>
```

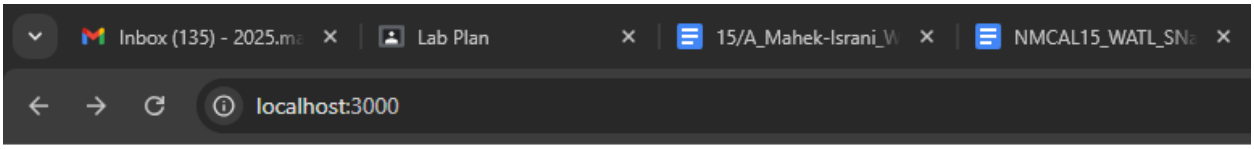
Contact.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Contact</title>
</head>
<body style="font-family:Arial;">
  <h1>Contact Information</h1>
  <p><b>Email:</b> johndoe@example.com</p>
  <p><b>Phone:</b> +91-9876543210</p>
```

```
  <p><b>Address:</b> My College, City, State, Country</p>
  <nav>
    <a href="/">Home</a> |
    <a href="/about">About Me</a>
  </nav>
</body>
</html>
```

Output:





Welcome to My College

Our college is a premier institution offering high-quality education across various departments.

Departments

- Computer Science
- Electronics
- Mechanical Engineering
- Civil Engineering

About Me | Contact

About Me

Hello! My name is **John Doe**, a student at My College.

I am passionate about web development and enjoy learning Node.js and backend technologies.

Home | Contact

Conclusion:

These tasks show how an HTTP server can deliver different file types, stream videos efficiently, and render multiple web pages using routing. Together, they provide a basic understanding of how web servers handle requests and serve content to users.

<table>
<tr><td colspan="4">Name of the student: Om Bhagwandas Wadhwani</td></tr>
<tr><td colspan="2">Roll No. 58/A</td><td colspan="2">Lab Practical No. 07</td></tr>
<tr><td colspan="4">Title of Lab Practical: Create an application to establish a connection with MySQL database and perform basic database operations on it.</td></tr>
<tr><td colspan="2">DOP: 28/10/2025</td><td colspan="2">DOS: 11/11/2025</td></tr>
<tr><td>CO Mapped: CO2</td><td>PO Mapped : PO1, PO3, PO4</td><td>Faculty Signature :</td><td>Marks:</td></tr>
</table>

Aim : To create a Node.js application that establishes a connection with a MySQL database (studentdb) and performs basic database operations such as inserting, updating, and deleting student records.

Description :

This practical shows how to use MySQL and Node.js to carry out simple database tasks. The MySQL server is connected to via the mysql2 module. The three primary database operations are covered in the practical:

- **INSERT:** Adding more than one student record.
- **UPDATE:** Changing a specific student's address.
- **DELETE:** Using a student's roll number to remove their record.

This aids students in comprehending how SQL queries and Node.js API techniques are used by backend apps to communicate with databases.

Problem Statement : Create a Node.js application that connects to a MySQL database named studentdb and performs the following operations on a table student(rollno, name, address):

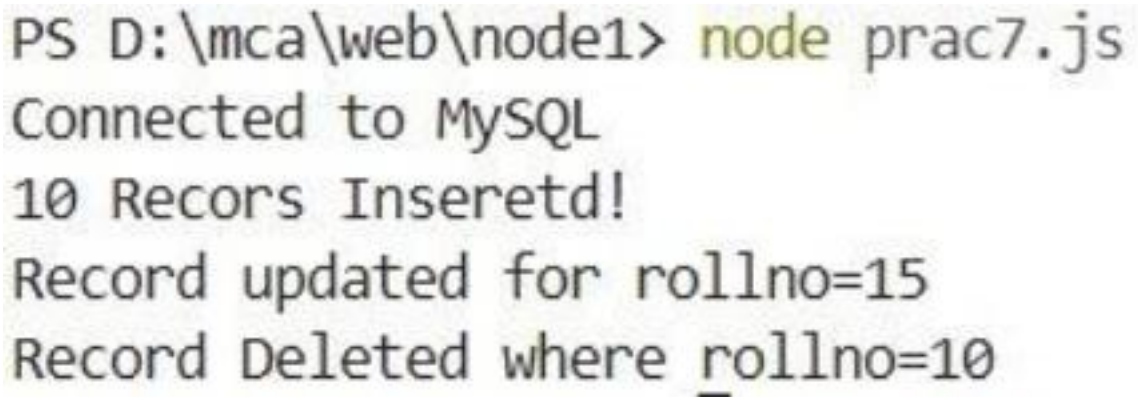
Code :

```
const mysql = require("mysql2");
const connection = mysql.createConnection({
 host: "localhost",
 user: "root",
 password: "root",
 database: "studentdb"
});

connection.connect((err) => {
 if (err) throw err;
 console.log("Connected to MySQL");
});
```

```
// Insert 10 records
let insertQuery = `
INSERT INTO student (rollno, name, address) VALUES
(11,'Karan','Rajastan'),
(12,'xyz','Delhi'),
(13,'PQR','Mumbai'),
(14,'Narendra','Haryana'),
(15,'Prathamesh','Mumbai'),
(16,'Om','Nashik'),
(17,'EFG','Hyderabad'),
(18,'Harsh','Bangalore'),
(19,'Ansh','Jaipur'),
(20,'Arun','Surat');
`;

connection.query(insertQuery, (err, result) => {
if (err) {
console.log("Records may already be inserted or some error
occurred.");
} else {
console.log("10 Records Inserted Successfully!");
}

// Update record
let updateQuery = "UPDATE student SET address='Mumbai'
WHERE rollno=15";
connection.query(updateQuery, (err, result) => {
if (err) throw err;
console.log("Record updated for rollno = 15");

// Delete record
let deleteQuery = "DELETE FROM student WHERE
rollno=10";  connection.query(deleteQuery, (err, result) => {
if (err) throw err;
```

```
console.log("Record deleted where rollno = 10");
connection.end();
});
});
});
```

Output :



Conclusion:

In this practical, we used the mysql2 module to successfully connect a Node.js application to a MySQL database. We carried the necessary database activities, such as adding several records, changing an existing record, and removing a record. This illustrates how SQL databases and backend apps work together to effectively handle data.

<table>
<tr><td colspan="4">Name of Student : Om Bhagwandas Wadhwani</td></tr>
<tr><td colspan="2">Roll Number : 58/A</td><td colspan="2">LAB Assignment Number:08</td></tr>
<tr><td colspan="4">Title of LAB Assignment : TypeScript installation, environment setup, and write programs based on decision-making, functions, and class & objects.</td></tr>
<tr><td colspan="2">DOP : 28/10/2025</td><td colspan="2">DOS : 11/11/2025</td></tr>
<tr><td>CO Mapped :
CO4</td><td>PO Mapped:
PO3,PO4,
PSO1,PSO2</td><td>Faculty Signature:</td><td>Marks :</td></tr>
</table>

Aim : To study and implement TypeScript installation, environment setup, and write programs based on decision-making, functions, and class & objects.

Description :

Type checking, interfaces, classes, and contemporary ES6+ compatibility are all characteristics of TypeScript, a strongly typed superset of JavaScript.

In this practical, we learn to:

- Install Node.js and TypeScript
- Set up TypeScript environment
- Compile & execute TypeScript programs
- Implement decision-making, functions, and class concepts

INSTALLATION & SETUP :

Step 1: Install Node.js

Download and install Node.js (LTS version) from the official website.

Step 2: Install TypeScript

```
I:\React>npm install -g typescript

added 1 package in 3s

I:\React>
```

Step 3: Verify Installation

```
I:\React>tsc -v
Version 5.9.3

I:\React>
```

Step 4: Create a TypeScript File

```
I:\React>tsc program.ts
```

Step 5 : Run JavaScript Output

```
I:\React>node program.js
Hello TypeScript

I:\React>
```

Problem Statement(i) : Write a TypeScript program to check whether a number is positive, negative, or zero.

Code :

```
let num: number = 10;
if (num > 0) {
 console.log(num + " is Positive");
} else if (num < 0) {
 console.log(num + " is Negative");
} else {
 console.log("Number is Zero");
}
```

Output :

```
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL> node practical8.ts
10 is Positive
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL>
```

Problem Statement(ii) : Write a TypeScript function to calculate the sum of two

numbers.

Code :

```
function addNumbers(a: number, b: number): number {

 return a + b;

}
let result = addNumbers(5, 7);
console.log("Sum = " + result);
```

Output :

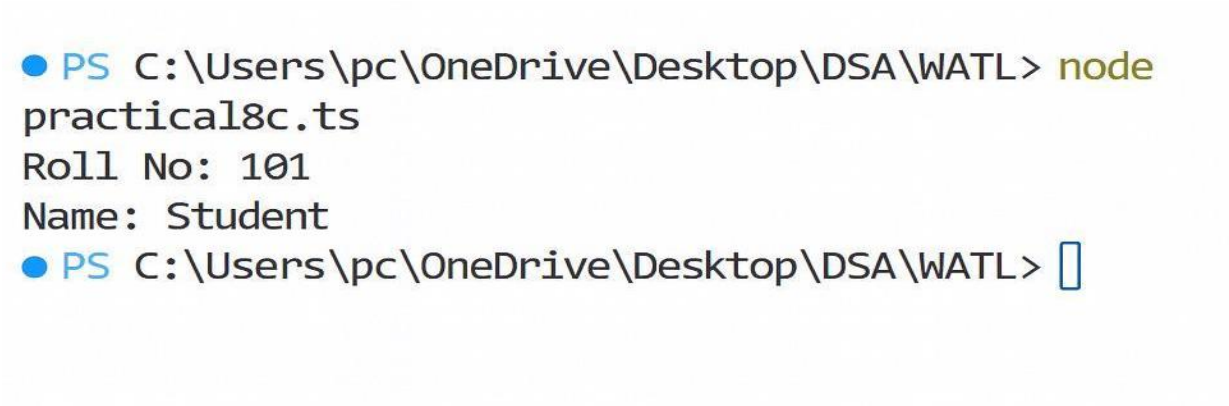
Problem Statement(iii) : Write a TypeScript program using Class & Object.

Code :

```
class Student {
 rollno: number;
 name: string;

 constructor(rollno: number, name: string) {
 this.rollno = rollno;
 this.name = name;
 }
```

```
display(): void {
console.log("Roll No: " + this.rollno);
console.log("Name: " + this.name);
}
}
let s1 = new Student(101, "student");
s1.display();
```

Output :



Conclusion:

Installing and configuring TypeScript, compiling.ts files, and implementing programs utilizing functions, class principles, and decision-making were all covered in this practical. This made it easier for us to comprehend how TypeScript gives JavaScript object-oriented functionality and type safety.

<table>
<tr><td colspan="4">Name of Student : Om Bhagwandas Wadhwani</td></tr>
<tr><td colspan="2">Roll Number : 58/A</td><td colspan="2">LAB Assignment Number:09</td></tr>
<tr><td colspan="4">Title of LAB Assignment :Introduction to Angular,setup for local development environment ,Angular Architecture</td></tr>
<tr><td colspan="2">DOP : 11/11/2025</td><td colspan="2">DOS :18/11/2025</td></tr>
<tr><td>CO Mapped :
CO5</td><td>PO Mapped:
PO3,PO4,
PSO1,PSO2</td><td>Faculty Signature:</td><td>Marks :</td></tr>
</table>

Aim : To study and understand Angular, its local development setup, Angular architecture, and create an application demonstrating the use of directives and pipes.

Description :

Angular is a framework for creating dynamic web apps that is based on TypeScript. This exercise covers the following topics:

- Introducing Angular
- Setting up an environment with Node.js and the Angular CLI
- Comprehending Angular architecture
- Developing an Angular application
- Showing off built-in pipes and directives

INSTALLATION & SETUP :

Step 1: Install Angular CLI

```
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL> npm install -g @angular/cli
>>

added 342 packages in 20s

71 packages are looking for funding
  run `npm fund` for details
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL>
```

```
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL> ng version
>>

Angular CLI       : 21.0.0
Node.js           : 24.8.0
Package Manager   : npm 11.6.0
Operating System  : win32 x64

PS C:\Users\pc\OneDrive\Desktop\DSA\WATL>
```

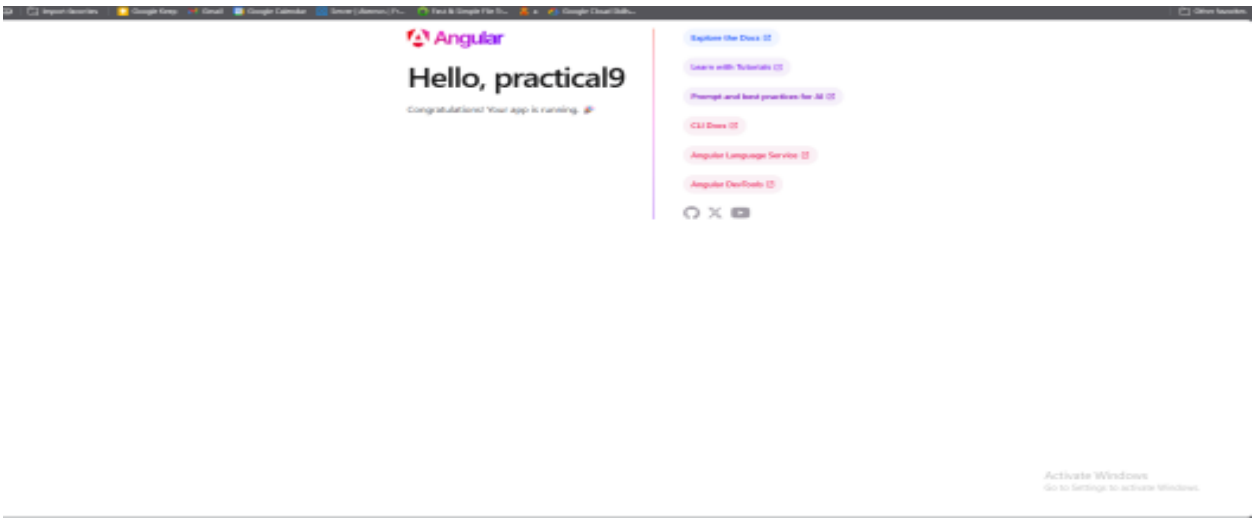
Step 3: Create an Angular Application

```
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL> ng new practical9
>>

Would you like to share pseudonymous usage data about this project with the Angular Team
at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more
details and how to change this setting, see https://angular.dev/cli/analytics.

   No
Global setting: disabled
Local setting: No local workspace configuration file.
Effective status: disabled
✓ Which stylesheet system would you like to use? Tailwind CSS    [ https://tailwindcss.com
✓ Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? Yes
✓ Which AI tools do you want to configure with Angular best practices? https://angular.dev/ai/develop-with-ai None
CREATE practical9/angular.json (2148 bytes)
CREATE practical9/package.json (1284 bytes)
CREATE practical9/README.md (1532 bytes)
CREATE practical9/tsconfig.json (990 bytes)
CREATE practical9/.editorconfig (331 bytes)
CREATE practical9/.gitignore (647 bytes)
```

```
CREATE practical9/src/app/app.html (20464 bytes)
CREATE practical9/src/app/app.config.ts (453 bytes)
CREATE practical9/src/app/app.routes.ts (80 bytes)
CREATE practical9/src/app/app.config.server.ts (438 bytes)
CREATE practical9/src/app/app.routes.server.ts (174 bytes)
CREATE practical9/public/favicon.ico (15086 bytes)
✓ Packages installed successfully.
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/styles.css', LF will be replaced by CRLF the next time Git touches it
    Successfully initialized git.
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL>
```

Step 4: Run the app:

```
>>
Error: This command is not available when running the Angular CLI outside a workspace.
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL> cd practical9
>>
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL\practical9> ng serve -o
>>
Browser bundles
Initial chunk files | Names         |  Raw size
main.js             | main          |  48.50 kB |
styles.css          | styles        |   6.03 kB |

                    | Initial total |  54.52 kB


Server bundles
Initial chunk files | Names          |  Raw size
main.server.mjs     | main.server    |  49.76 kB |
server.mjs          | server         |   1.81 kB |
polyfills.server.mjs | polyfills.server | 243 bytes |
```



Problem Statement(ii) : Create an Angular application demonstrating Pipes.

Code (app.html)

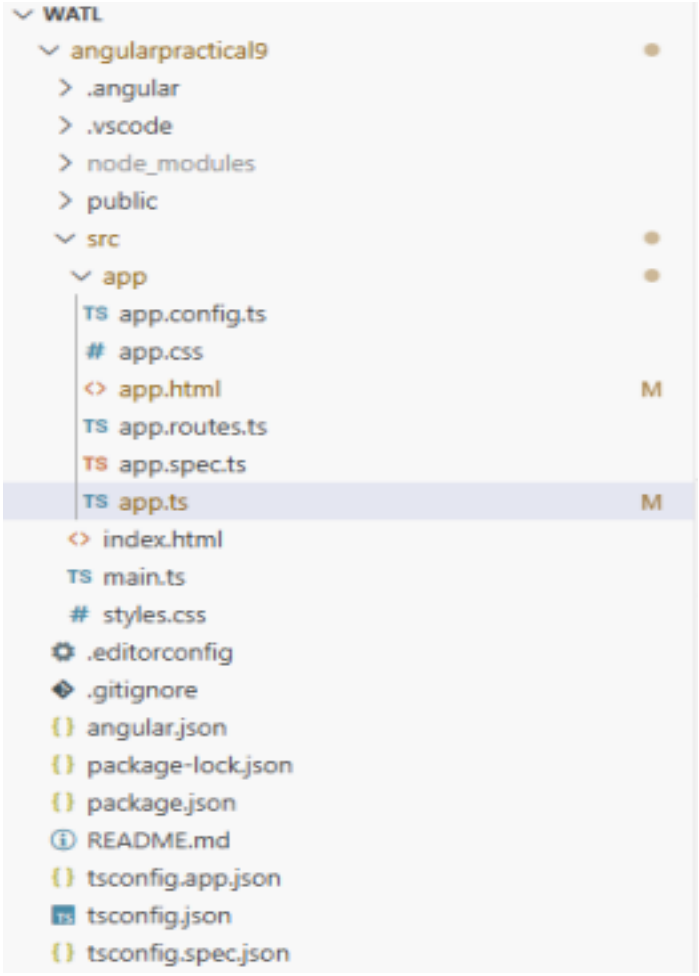
```
    <h2>Angular Directives Demo</h2>
<!-- ngIf -->
<p *ngIf="isShown">This text is shown using *ngIf directive</p>
<!-- ngFor -->
<ul>
 <li *ngFor="let item of fruits">{{ item }}</li>
</ul>
<!-- ngStyle -->
<p [ngStyle]="{'color':'blue', 'font-size':'20px'}">
 This text is styled using ngStyle directive
</p>
```

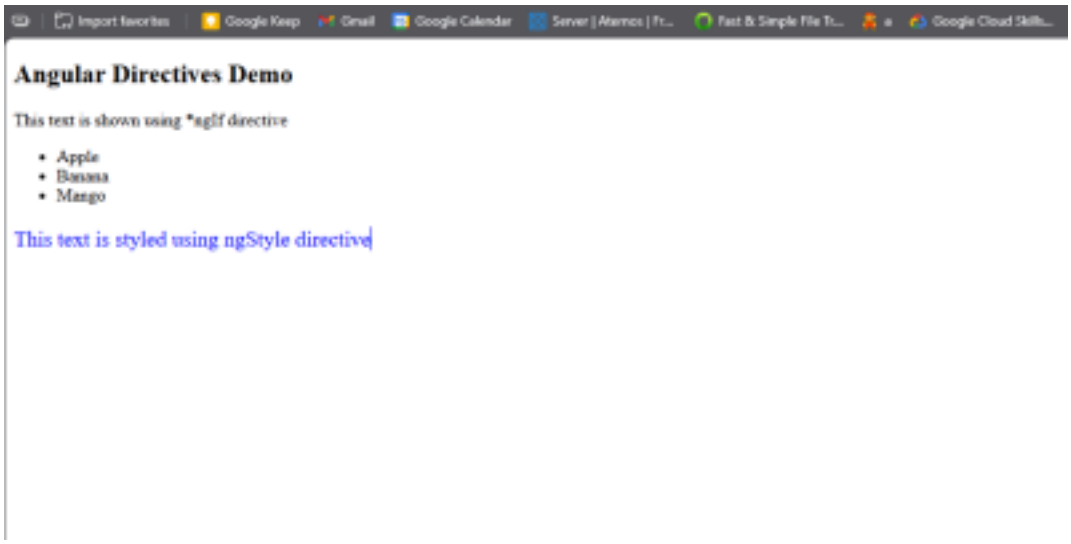
Code (app.ts)

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
```

```
@Component({
 selector: 'app-root',
 standalone: true,
 imports: [CommonModule],
 templateUrl: './app.html',
 styleUrl: './app.css'
})
export class App {
 isShown = true;
 fruits = ['Apple', 'Banana', 'Mango'];
}
```

Output :





Problem Statement(ii) : Create an Angular application demonstrating

Pipes.

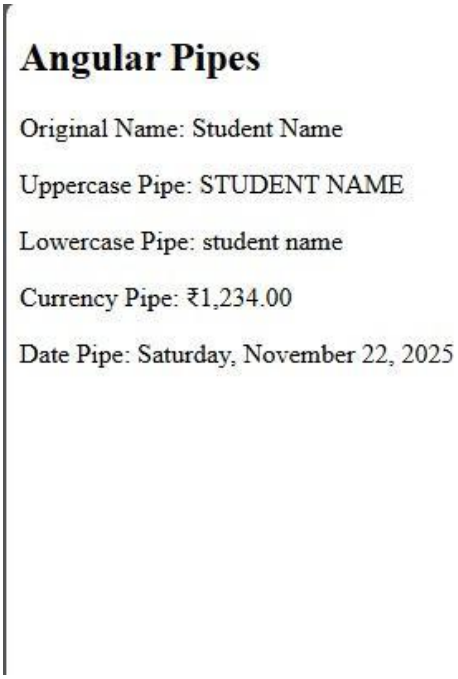
Code (app.html)

```
<h2>Angular Pipes</h2>

<p>Original Name: {{ name }}</p>
<p>Uppercase Pipe: {{ name | uppercase }}</p>
<p>Lowercase Pipe: {{ name | lowercase }}</p>
<p>Currency Pipe: {{ price | currency:'INR' }}</p>
<p>Date Pipe: {{ today | date:'fullDate' }}</p>
```

Code (app.ts)

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';

@Component({
 selector: 'app-root',
 standalone: true,
 imports: [CommonModule],
 templateUrl: './app.html',
 styleUrl: './app.css'
})
export class App {
 name = "STUDENT NAME";
 price = 1,234;
 today = new Date();
}
```

Output :



Conclusion:

We successfully configured the Angular development environment, comprehended the Angular architecture, and produced an Angular project during this practical. In order to comprehend how Angular directives (ngIf, ngFor, ngStyle) and pipes (uppercase, lowercase, currency, date) are used within components, we also constructed and illustrated their use.

<table>
<tr><td colspan="4">Name of Student : Om Bhagwandas Wadhwani</td></tr>
<tr><td colspan="2">Roll Number : 58/A</td><td colspan="2">LAB Assignment Number:10</td></tr>
<tr><td colspan="4">Title of LAB Assignment : Demonstrate features of Angular forms with a program</td></tr>
<tr><td colspan="2">DOP : 18/11/2025</td><td colspan="2">DOS :26/11/2025</td></tr>
<tr><td>CO Mapped :
CO4</td><td>PO Mapped:
PO3,PO4,
PSO1,PSO2</td><td>Faculty Signature:</td><td>Marks :</td></tr>
</table>

Aim : Demonstrate features of Angular forms with a program

Description :

User input, validation, error handling, and submission are all made simple with Angular Forms.

We employ Template-Driven Forms in this exercise, which rely on Angular directives like:

- ngForm
- ngModel
- required
- minlength
- email

Problem Statement : Create an Angular application to demonstrate the use of Angular template-driven forms, including form creation, validation, and submission.

Code (app.html)

```
<h2>Angular Template-Driven Form</h2>
<form #studentForm="ngForm" (ngSubmit)="onSubmit()">
 <label>Name:</label>
 <input
 type="text"
 name="name"
 [(ngModel)]="student.name"
 required
 minlength="3"
 #name="ngModel">
 <div class="error" *ngIf="name.invalid && name.touched">
 Name is required (min 3 characters)
 </div>
 <br>
 <label>Email:</label>
 <input
type="email"
 name="email"
 [(ngModel)]="student.email"
 required
```

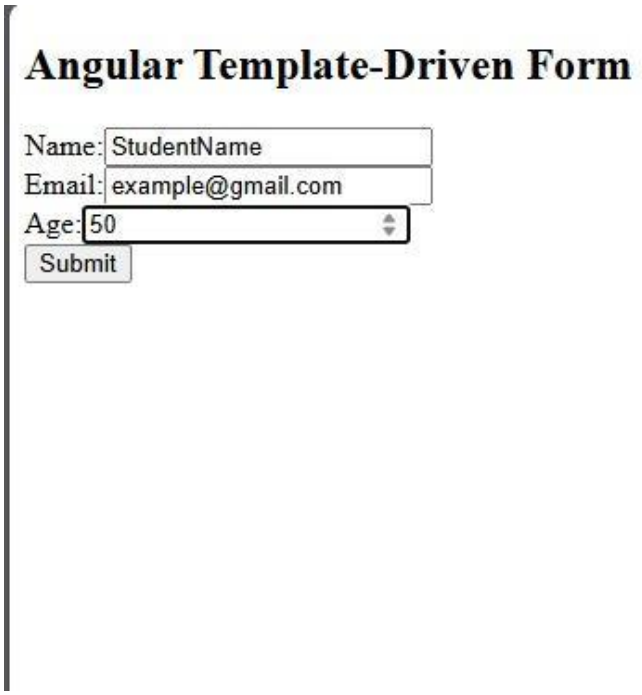
```
email
#email="ngModel">
<div class="error" *ngIf="email.invalid && email.touched">
Enter a valid email
</div>
<br>
<label>Age:</label>
<input
type="number"
name="age"
[(ngModel)]="student.age"
required
min="1"
#age="ngModel">
<div class="error" *ngIf="age.invalid && age.touched">
Age must be above 0
</div>
<br>
<button type="submit" [disabled]="studentForm.invalid">
Submit
</button>
</form>
<div *ngIf="submitted">
<h3>Submitted Data:</h3>
<p><strong>Name:</strong> {{ student.name }}</p>
<p><strong>Email:</strong> {{ student.email }}</p>
<p><strong>Age:</strong> {{ student.age }}</p>
</div>
```

Code (app.ts)

```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';

@Component({
 selector: 'app-root',
 standalone: true,
 imports: [FormsModule],
```

```
templateUrl: './app.html',
styleUrl: './app.css'
})
export class App {
student = {
name: '',
email: '',
age: null
};

submitted = false;

onSubmit() {
this.submitted = true;
}
}
```

Output :



Conclusion:

Using ngModel, ngForm, and Angular validation characteristics, this practical effectively illustrates Angular's Template-Driven Forms, including input binding, validation, form state checking, and form submission.

<table>
<tr><td colspan="4">Name of Student : Om Bhagwandas Wadhwani</td></tr>
<tr><td colspan="2">Roll Number :58/A</td><td colspan="2">LAB Assignment Number: 11</td></tr>
<tr><td colspan="4">Title of LAB Assignment : Create an application to demonstrate SPA</td></tr>
<tr><td colspan="2">DOP :26/11/2025</td><td colspan="2">DOS :03/12/2025</td></tr>
<tr><td>CO Mapped: CO6</td><td>POMapped: PO3,PO4, PSO1,PSO2</td><td>Faculty Signature:</td><td>Marks :</td></tr>
</table>

Aim : Create an application to demonstrate SPA

Description :

A Single Page Application (SPA) loads a single HTML page and dynamically updates the content as the user interacts.

Angular provides a powerful routing module to create SPAs easily.

In this practical, we create three components (Home, About, Contact) and configure Angular routes so the application behaves as a SPA.

INSTALLATION & SETUP :

Step 1: Create a new Angular project
Step 2: Run the project
IStep 3: Create Components

```
→ Press h + enter to show help
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL\angularpractical11> ng generate component home
CREATE src/app/home/home.spec.ts (540 bytes)
CREATE src/app/home/home.ts (189 bytes)
CREATE src/app/home/home.css (0 bytes)
CREATE src/app/home/home.html (20 bytes)
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL\angularpractical11> ng generate component about
CREATE src/app/about/about.spec.ts (547 bytes)
CREATE src/app/about/about.ts (193 bytes)
CREATE src/app/about/about.css (0 bytes)
CREATE src/app/about/about.html (21 bytes)
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL\angularpractical11> ng generate component contact
CREATE src/app/contact/contact.spec.ts (561 bytes)
CREATE src/app/contact/contact.ts (201 bytes)
CREATE src/app/contact/contact.css (0 bytes)
CREATE src/app/contact/contact.html (23 bytes)
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL\angularpractical11> ng serve --open
```

Problem Statement : Create an Angular application demonstrating SPA (Single Page Application) using routing and navigation between components.

Code app.routes.ts :

```
import { Routes } from
'@angular/router'; import { Home }
from './home/home';

import { About } from './about/about';

import { Contact } from './contact/contact';

export const routes: Routes = [

  { path: '', component: Home },

  { path: 'about', component: About },

  { path: 'contact', component: Contact }

];
```

Code app.html :

```
<h1>Angular SPA Demo</h1>

  <nav>

    <a routerLink="">Home</a> |

    <a routerLink="about">About</a> |

    <a routerLink="contact">Contact</a>

  </nav>
```

```
<hr>

<router-outlet></router-outlet>
```

Code home.html :

```
<h2>Home Page</h2>

<p>This is the Home component</p>
```

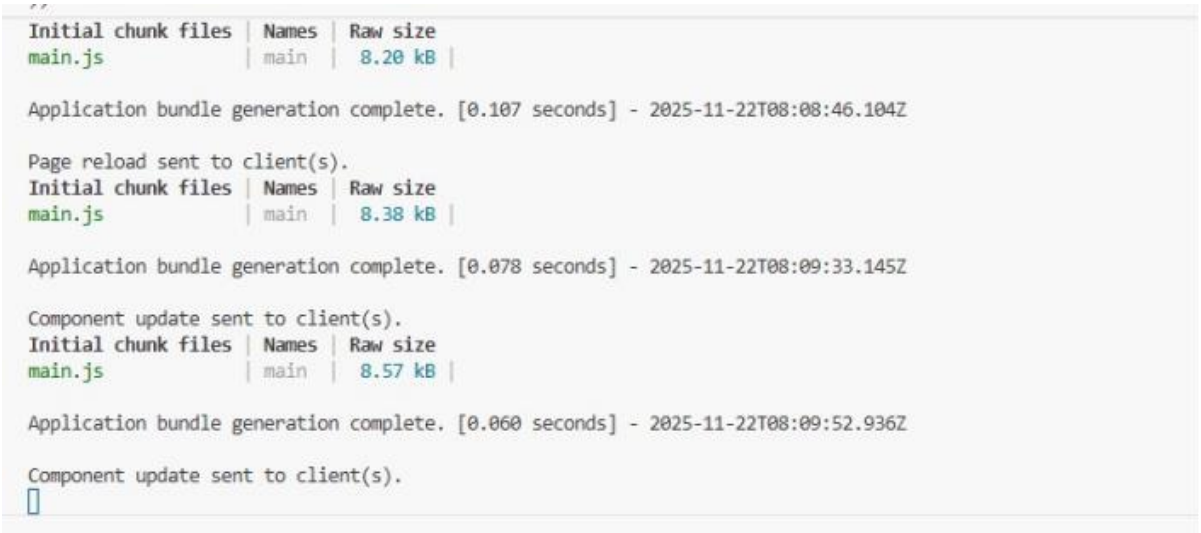
Code about.html :

```
<h2>About Page</h2>

<p>This is the About component</p>
```

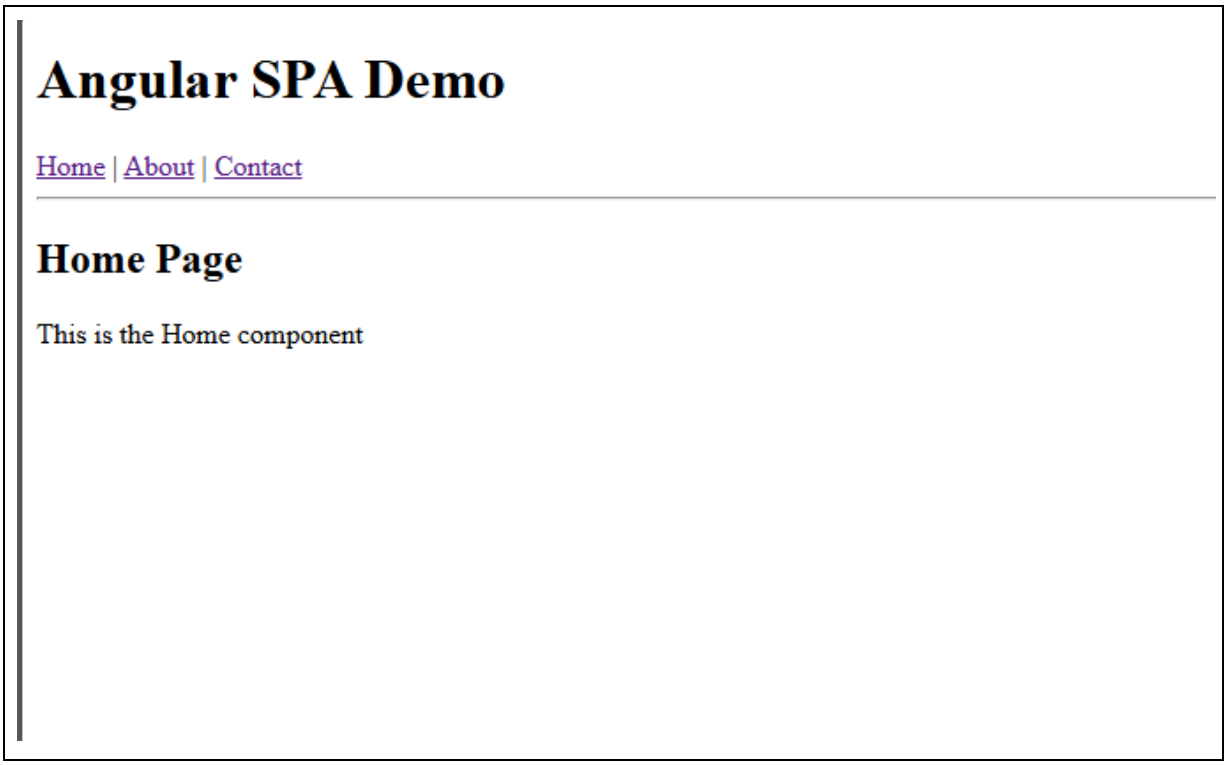
Code contact.html :

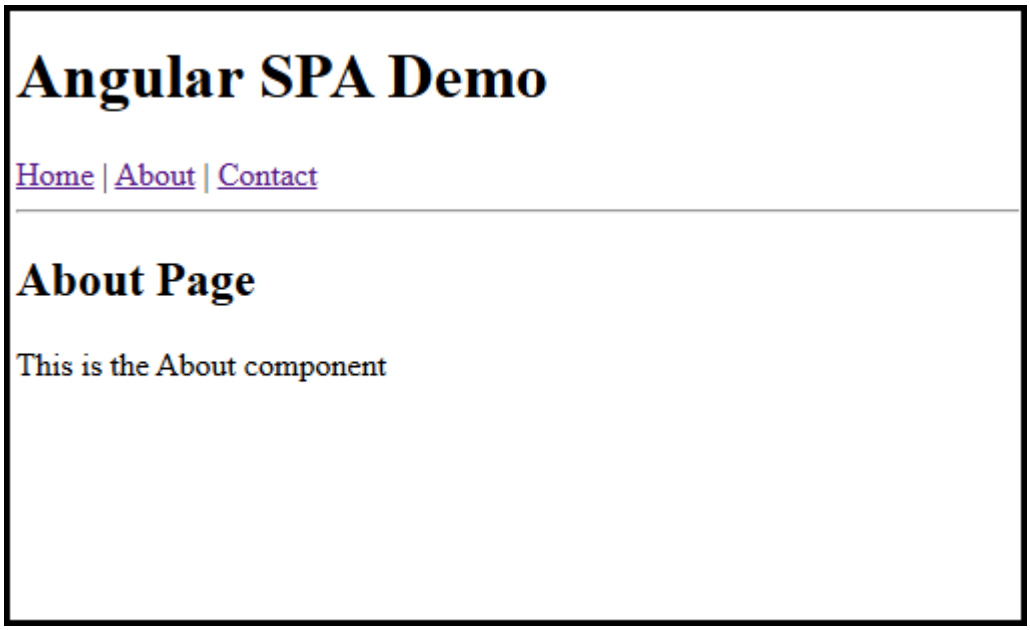
```
<h2>Contact Page</h2>

<p>This is the Contact component</p>
```

Output :



```
Initial chunk files | Names | Raw size
main.js             | main  |  8.20 kB |

Application bundle generation complete. [0.107 seconds] - 2025-11-22T08:08:46.104Z

Page reload sent to client(s).
Initial chunk files | Names | Raw size
main.js             | main  |  8.38 kB |

Application bundle generation complete. [0.078 seconds] - 2025-11-22T08:09:33.145Z

Component update sent to client(s).
Initial chunk files | Names | Raw size
main.js             | main  |  8.57 kB |

Application bundle generation complete. [0.060 seconds] - 2025-11-22T08:09:52.936Z

Component update sent to client(s).
```


Angular SPA Demo
Home | About | Contact
Home Page
This is the Home component



Conclusion:

In this practical, we successfully created an Angular Single Page Application using Angular Routing.

- We demonstrated:
- Navigation between pages
- Use of routerLink
- Dynamic content loading
- No page reload

This fulfills the requirements of an SPA using Angula

ASSIGNMENT – 1

Aim : To study and implement basic Node.js concepts including synchronous vs asynchronous execution, event-driven modules using EventEmitters, and HTTP server creation with file handling.

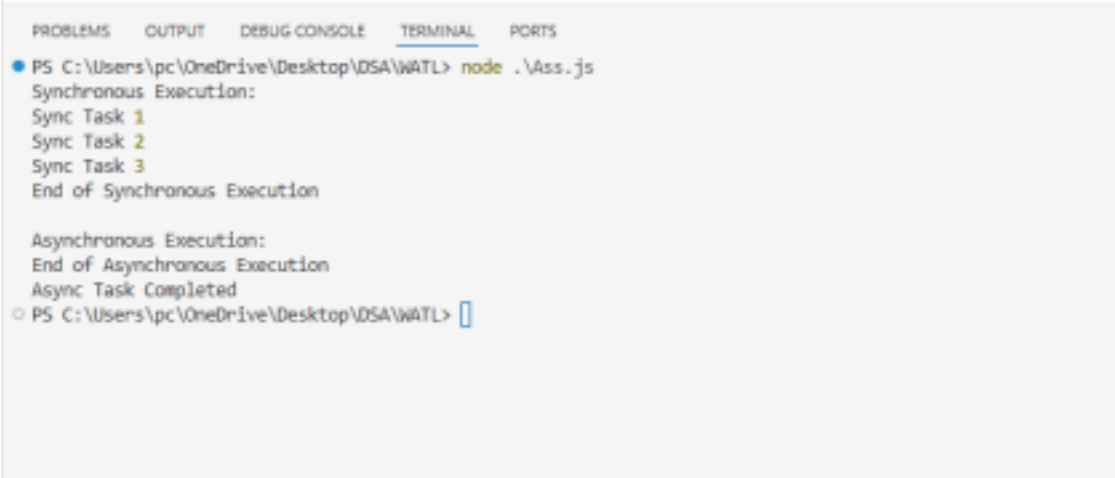
Description :

Three key Node.js functionalities are demonstrated in this practical.
To demonstrate how Node.js manages blocking and non-blocking tasks, it first contrasts synchronous and asynchronous execution.
Second, it highlights event-driven architecture by implementing a custom module that releases an event each time a function is carried out.
In order to demonstrate server development and file system interaction, it finally builds an HTTP server that reads the contents of a text file and transmits them to the client.

Problem Statement 1: Program to Demonstrate Synchronous vs Asynchronous Execution Code :

```
console.log("Synchronous Execution:");
function syncTask() {
 for (let i = 1; i <= 3; i++) {
 console.log("Sync Task", i);
 }
 }
syncTask();
console.log("End of Synchronous Execution\n");
console.log("Asynchronous Execution:");
function asyncTask() {
 setTimeout(() => {
 console.log("Async Task Completed");
 }, 2000);
 }
asyncTask();
console.log("End of Asynchronous Execution");
```

Output :



Problem Statement 2: Program to Create a Module That Emits an Event

Code :

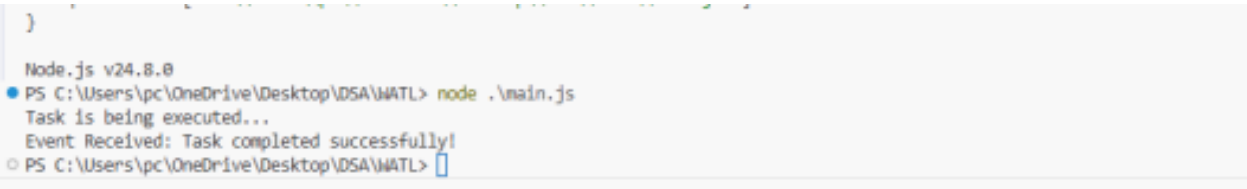
File 1: eventModule.js

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();

function executeTask() {
 console.log("Task is being executed...");
 myEmitter.emit('taskDone');
}

module.exports = { myEmitter, executeTask };
```

File 2: main.js

```
const { myEmitter, executeTask } = require('./eventModule');

myEmitter.on('taskDone', () => {
 console.log("Event Received: Task completed
 successfully!"); });

executeTask();
```

Output :



Problem Statement 3: Create an HTTP Server That Reads a Text File and Sends It as a Response

Code :

File 1: httpfile.js

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

const server = http.createServer((req, res) => {
 fs.readFile('sample.txt', 'utf8', (err, data) => {
 if (err) {
 res.writeHead(500, {'Content-Type': 'text/plain'});
 return res.end("Error reading file");
 }
 res.writeHead(200, {'Content-Type': 'text/plain'});
 res.end(data);
 });
});

server.listen(3000, () => {
 console.log("Server is running on http://localhost:3000");
});
```

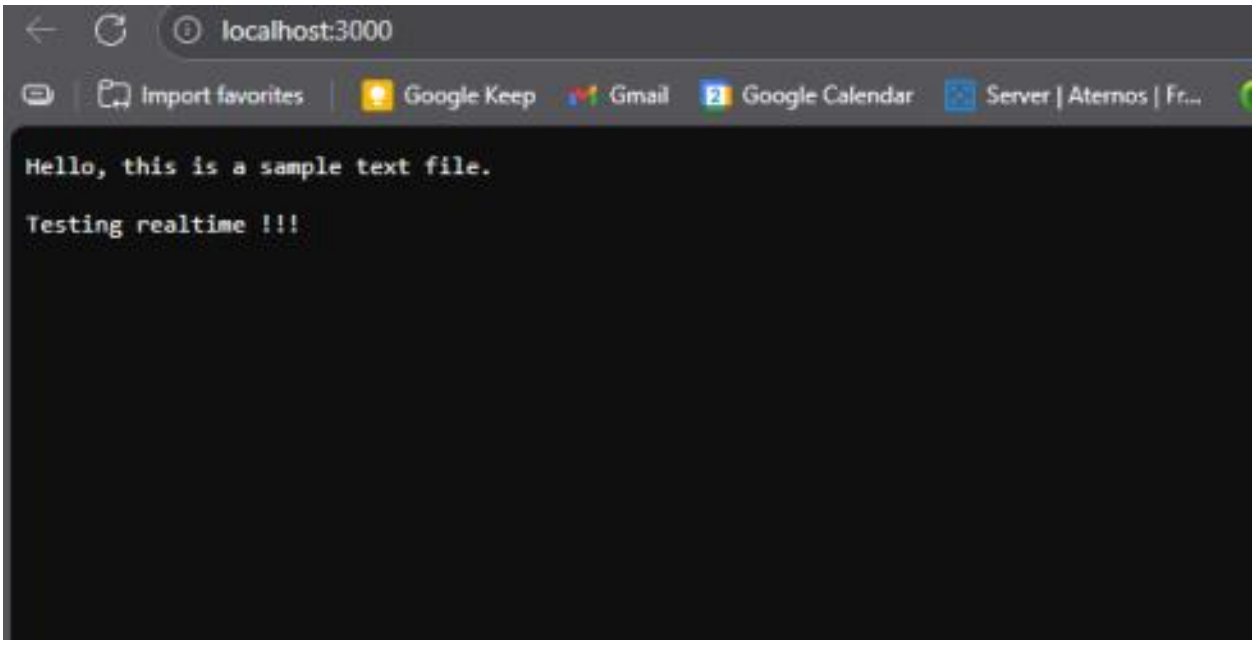
File 2: sample.txt

Hello, this is a sample text file.

Testing realtime !!!

Output :

```
PS C:\Users\pc\OneDrive\Desktop\DSA\WATL> node .\assc.js
Server is running on http://localhost:3000
```



Conclusion:

We investigated important aspects of Node.js with this project, such as server-side file processing with an HTTP server, event-driven programming with custom modules, and synchronous vs. asynchronous execution. These courses aid in developing a solid foundation in Node.js server operations, concurrency, and modularity—all crucial abilities for backend development.

ASSIGNMENT – 2

Aim : Modularization and Classes

Description :

Three important facets of contemporary application development are covered in this assignment. The first program uses Node.js and MySQL to extract and print every record from a database table called students.
The second program uses a loop to print every day of the week as an example of iteration. In order to demonstrate the Angular dependency injection method, the third program creates a service (UserService) and injects it into a component.

Q1) : Fetch All Records from Students Table

Code :

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

const connection = mysql.createConnection({
 host: "localhost",
 user: "root",
 password: "root",
 database: "college"
});

connection.connect((err) => {
 if (err) throw err;
 console.log("Connected to MySQL database!");

 connection.query("SELECT * FROM students", (err,
results) => {  if (err) throw err;
 console.log("Students Table Records:");
 console.log(results);
 connection.end();
 });
```

```
});
```

Output :

```
Node.js v20.18.0
PS C:\Users\user\Desktop\Tanay> node .\assognment2
Connected to MySQL database!
Students Table Records:
[
  RowDataPacket { id: 1, name: 'Amit', dept: 'IT', marks: 85 },
  RowDataPacket { id: 2, name: 'Priya', dept: 'CS', marks: 90 },
  RowDataPacket { id: 3, name: 'Rohan', dept: 'EXTC', marks: 78 }
]
```

Q2) : Display Days of the Week Using a Loop

Code :

File 1: eventModule.js

```
const days = [
 "Monday", "Tuesday", "Wednesday",
 "Thursday", "Friday", "Saturday", "Sunday"
];

console.log("Days of the Week:");
for (let day of days) {
 console.log(day);
}
```

Output :

```
PS C:\Users\user\Desktop\Tanay> node .\ass2b.js
Days of the Week:
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
PS C:\Users\user\Desktop\Tanay>
```

Q3) : Angular UserService and Inject into Component

Code :

File 1: user.service

```
import { Injectable } from '@angular/core';

@Injectable({
 providedIn: 'root'
})
export class UserService {

 getUser() {
 return {
 id: 1,
 name: "John Doe",
 email: "john@example.com"
 };
 }
}
```

File 2: app.component

```
import { Component } from '@angular/core';
import { UserService } from './user.service';

@Component({
 selector: 'app-root',
 template: `
 <h2>User Details</h2>
 <p>ID: {{ user.id }}</p>
 <p>Name: {{ user.name }}</p>
 <p>Email: {{ user.email }}</p>
```

```
`
})
export class AppComponent {
 user: any;

 constructor(private userService: UserService) {
 this.user = this.userService.getUser();
 }
 }
```

Output :

```
User Details
ID: 1
Name: John Doe
Email: john@example.com
PS C:\Users\user\Desktop\Tanay>
```

Conclusion:

This assignment taught us how to use Node.js for backend database operations, practiced JavaScript looping structures by listing every day of the week, and investigated Angular's dependency injection by building and utilizing a custom service. Both server-side and front-end application development skills are strengthened by these subjects.