Roll No. 45 Exam Seat No.

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Hashu Advani Memorial Complex, Collector's Colony, R. C. Marg,
Chembur, Mumbai – 400074. Contact No. 02261532532

V.E.S.

Since 1962

CERTIFICATE

Certified that Mr. Pushkar Prasad Sane

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 20 23 – 2024.

Principal Head of Department

Lab In-charge Subject Teacher

V.E.S. Institute of Technology, Collector Colony,
Chembur, Mumbai, Maharashtra 400047
Department of M.C.A

INDEX

Sr. No.	Contents	Date of Preparation	Date of Submission	Marks	Faculty Sign
1	**Practical 1** 1. Introduction to Node.JS, Advantages and Disadvantages,Node.js Process Model, Traditional Web Server Model, Installation. 2. To print today's date and time using REPL. 3. Write a program to print the given pattern: 1 2 3 4 5 1 2 3 4 1 2 3 1 2 1 4. Write a program to print the first 20 Fibonacci numbers (take input through the command line). 5. Write a program to print prime nos between 1 to 100.	12-09-2023	12-09-2023		

2	**Practical 2** 1. Create an application to demonstrate Node.js Modules. 2. Write a program to print information about the computer's operating system using the OS module(use any 5 methods). 3. Print "Hello" every 500 milliseconds using the Timer Module. The message should be printed exactly 10 times. Use SetInterval ,ClearInterval and SetTimeout methods. 4. Create a Calculator Node.js Module with functions add, subtract and multiply,Divide and use the Calculator module in another Node.js file. 5. Create a circle module with functions to find the area and perimeter of a circle and use it.	12-09-2023	13-09-2023		
3	**Practical 3** 1. Create an application to demonstrate various Node.js Events in event emitter class. 2. Create functions to sort, reverse and search for an element in an array. Register	25-09-2023	26-09-2023		

4	and trigger these functions using events.				
4	**Practical 4** ● Create an application to demonstrate Node.js Functions- Timer function(displays every 10 second)	26-09-2023	28-09-2023		
5	**Practical 5** ● Using File Handling demonstrate all basic file operations (Create, write, read, delete). 1. Read a file. 2. Write to an existing file. 3. Create a file. 4. Delete a file.	09-10-2023	10-10-2023		
6	**Practical 6** 1. Create a HTTP Server and serve HTML, CSV, JSON and PDF Files. 2. Create a HTTP Server and stream a video file using piping. 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.	11-10-2023	13-10-2023		

7	**Practical 7** ● Create an application to establish a connection with the MySQL database and perform basic database operations on it(student db consisting roll no, name, address), insert 10 records, update a particular student's record,delete a record.	16-10-2023	17-10-2023		
8	**Practical 8** 1. TypeScript installation, Environment Setup, Programs on decision making / functions / class & object) 2. Programs on decision making / functions / class & object)	16-10-2023	20-10-2023		
9	**Practical 9** 1. Introduction to Angular,Setup for local Development environment, Angular Architecture. 2. Create an application to demonstrate directives and pipes.	22-09-2023	23-09-2023		
10	**Practical 10** ● Create an application to demonstrate directives and pipes.	23-09-2023	24-09-2023		

11	**Practical 11** ● Create an application to demonstrate SPA.	24-10-2023	24-10-2023		

Final Grade	Instructor Signature

Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 1**	
Title of Lab Assignment: Understand the concepts of REPL and Node.js console.			
DOP: 12-09-2023		**DOS: 12-09-2023**	
CO Mapped: **CO1**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Signature:**	**Marks:**

Practical 1

Aim:

1) To understand the fundamentals of Node.js, its advantages, disadvantages, and key concepts like the Node.js process model and traditional web server model Installation.
2) Print Today's date and time using REPL.
3) Write a program to Print the given pattern:

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

4) Write a Program to print the first 20 Fibonacci numbers (Take input through the command line).
5) Write a program to print prime nos between 1 to 100.

Theory:

1) **Introduction to Node.js**

   Node.js, an open-source and cross-platform JavaScript runtime environment, is constructed upon Google Chrome's V8 JavaScript engine. Its primary purpose is to facilitate the execution of JavaScript code beyond the confines of web browsers, rendering it a robust choice for server-side development. Node.js boasts an event-driven, non-blocking architecture, which endows it with the capability to efficiently manage numerous concurrent connections. These attributes make it a formidable player in modern web development.

2) **Advantages of Node.js:**

   Highly Efficient: Node.js employs an event-driven, non-blocking I/O model, a hallmark of its efficiency in managing concurrent requests. This model ensures that Node.js can adeptly handle multiple tasks simultaneously without performance bottlenecks.

Single Language: One of Node.js' remarkable strengths is that it allows developers to utilize JavaScript for both client-side and server-side scripting. This unification of programming languages streamlines the development process and eliminates the need to switch between languages.

Vast Ecosystem: Node.js boasts an extensive ecosystem of libraries and packages available through npm (Node Package Manager), simplifying development by providing readily accessible resources.

Scalability: Designed with scalability in mind, Node.js excels at accommodating a substantial number of concurrent connections, making it an excellent choice for applications that expect high traffic.

Real-time Applications: Node.js is the ideal platform for crafting real-time applications like chat applications, online gaming platforms, and live data streaming services, thanks to its event-driven, non-blocking nature.

3) Disadvantages of Node.js:

Single-threaded: Node.js operates on a single-threaded event loop, which can lead to performance issues for CPU-bound tasks, limiting its suitability for certain types of applications.

Callback Hell: Managing asynchronous code using callbacks can result in a phenomenon known as "callback hell," where code becomes complex and challenging to read, potentially hampering development efficiency.

Lack of In-built Modules: Some modules available in other programming languages may not have readily available counterparts in the Node.js ecosystem, necessitating additional development efforts.

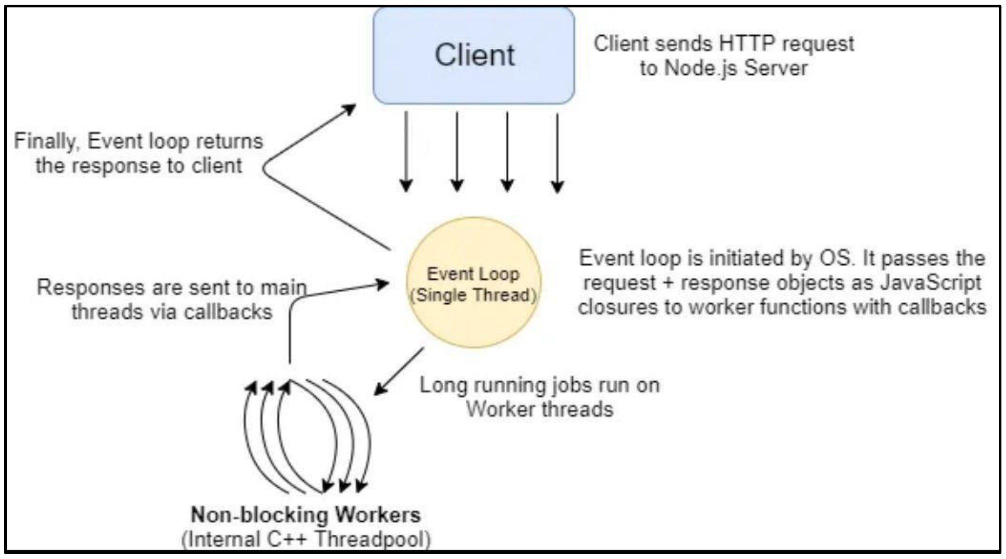
Less Suitable for Large-scale Applications: While Node.js is highly capable, it may not be the best choice for very large and intricate applications that require complex orchestration and multi-threaded processing.

4) Node.js Process Model:

Node.js adopts a unique process model that distinguishes it from traditional web servers. In Node.js, all code runs within a single process, and requests are handled on a single thread. This approach offers several advantages, including reduced resource consumption and improved scalability.
When a request arrives in a Node.js application, it doesn't block the main thread. Instead, it's placed in an event queue. Node.js employs an event loop, a critical component, to continuously monitor this queue for events, such as incoming requests or completed asynchronous operations. This event loop efficiently manages the flow of requests and ensures that the server remains responsive.

Non-Blocking Requests: These are requests that don't involve time-consuming computations or data retrievals. In such cases, Node.js can immediately process the request, construct the response, and send it back to the client without waiting.

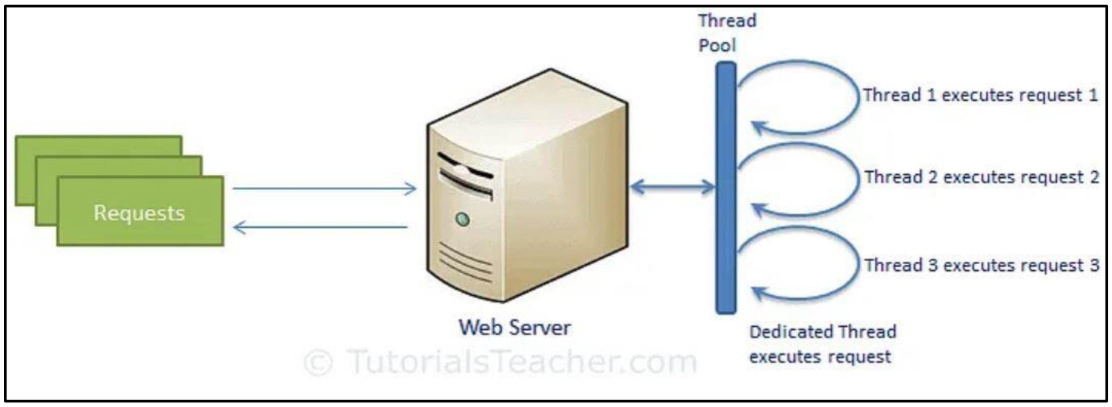
Blocking Requests: Some requests require I/O operations, like reading from a file or querying a database. These operations can take a significant amount of time. In Node.js, blocking requests are offloaded to a worker thread pool. Each request sent to the pool is associated with a callback function, which is executed when the operation completes. This means that while one thread is waiting for I/O to finish, the main thread can continue processing other requests, making the most of the available resources.


Client
Client sends HTTP request to Node.js Server
Finally, Event loop returns the response to client
Event Loop (Single Thread)
Event loop is initiated by OS. It passes the request + response objects as JavaScript closures to worker functions with callbacks
Responses are sent to main threads via callbacks
Long running jobs run on Worker threads
Non-blocking Workers (Internal C++ Threadpool)

5) Web Server Model:

Imagine a traditional web server as a team of workers. Each worker (or thread) can handle one task at a time. When a new task (or request) arrives, it gets assigned to an available worker. However, if all the workers are currently busy with other tasks, the new task must wait its turn. It's like waiting in line at a busy store; you can't be helped until the cashier is available.

This way of handling requests is called synchronous or blocking because tasks are processed in a sequential, step-by-step fashion. It's like a single lane road where cars must wait for the one in front to move before they can proceed. This approach ensures that requests are processed in order but can sometimes result in delays if there are many tasks or if some tasks take a long time to complete.


Thread Pool
Requests
Thread 1 executes request 1
Thread 2 executes request 2
Thread 3 executes request 3
Web Server
© TutorialsTeacher.com
Dedicated Thread executes request

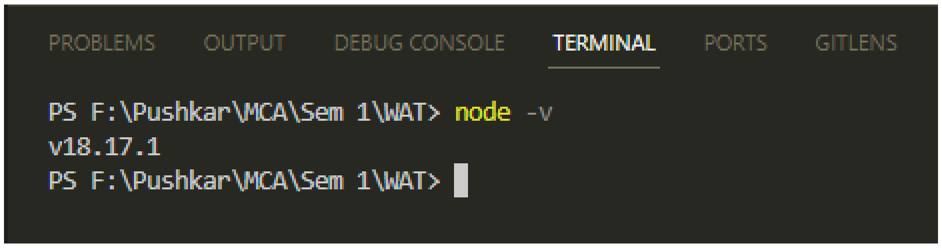
6) Installation:

Step 1: Downloading the Node.js '.msi' installer.

Step 2: Running the Node.js installer.

Step 3: Check Node.js and NPM Version

Command: node -v



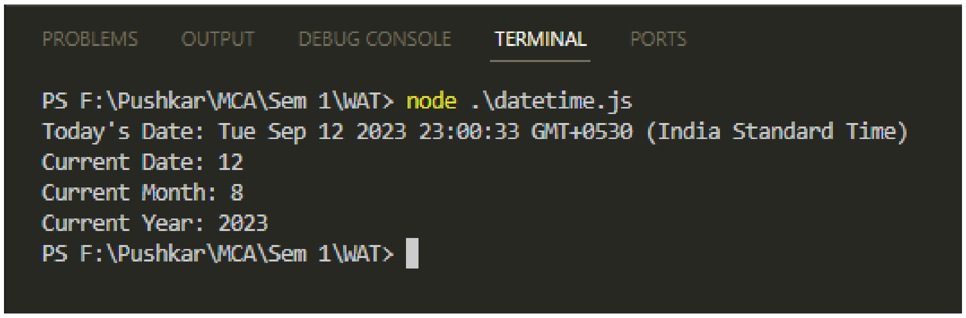
```
PS F:\Pushkar\MCA\Sem 1\WAT> node -v
v18.17.1
PS F:\Pushkar\MCA\Sem 1\WAT>
```

1. **Print today's date and time using REPL.**

Code:

```
const today = new Date();
console.log("Today's Date: " + today);
console.log("Current Date: " + today.getDate());
console.log("Current Month: " + today.getMonth());
console.log("Current Year: " + today.getFullYear());
```

Output:



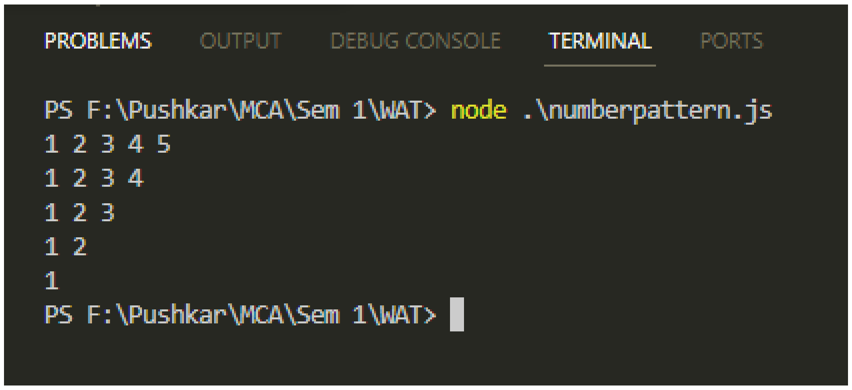
```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\datetime.js
Today's Date: Tue Sep 12 2023 23:00:33 GMT+0530 (India Standard Time)
Current Date: 12
Current Month: 8
Current Year: 2023
PS F:\Pushkar\MCA\Sem 1\WAT>
```

2. **Write a program to print a given pattern.**

Code:

```
function printPattern(rows) {
    for (let i = rows; i >= 1; i--) {
        let pattern = '';
        for (let j = 1; j <= i; j++) {
            pattern += j + ' ';
        }
        console.log(pattern);
    }
}
const numRows = 5;
printPattern(numRows);
```

Output:



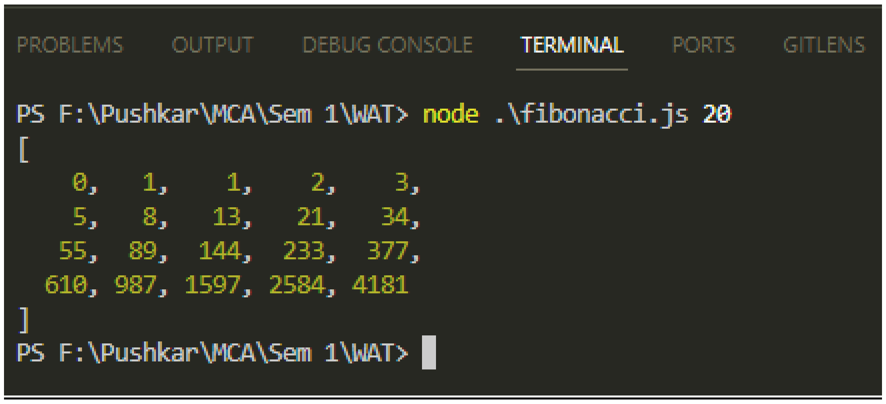
```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\numberpattern.js
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
PS F:\Pushkar\MCA\Sem 1\WAT>
```

3. **Write a program to print the first 20 fibonacci numbers (take input through command line).**

Code:

```
const n = parseInt(process.argv[2]);
function fibonacci(n){
   let fib = [0, 1];
   for(let i = 2; i < n; i++){
      fib[i] = fib[i - 1] + fib [i - 2];
   }
   return fib;
}
console.log(fibonacci(n));
```

Output:



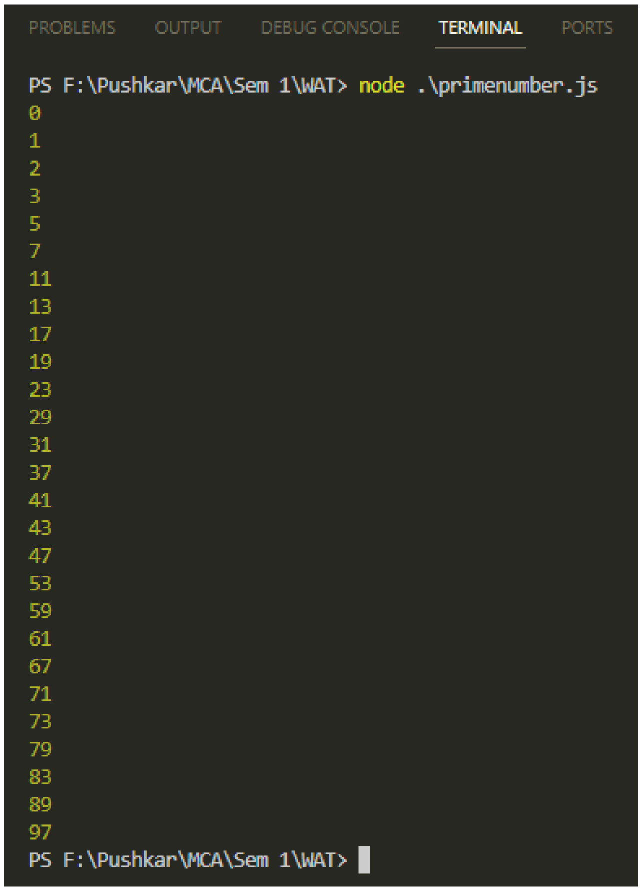
```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\fibonacci.js 20
[
    0,   1,    1,    2,    3,
    5,   8,   13,   21,   34,
   55,  89,  144,  233,  377,
  610, 987, 1597, 2584, 4181
]
PS F:\Pushkar\MCA\Sem 1\WAT>
```

4. Write a program to print numbers between 0 and 100.

Code:

```
for (var num = 0; num <= 100; num++) {
    var notPrime = false;
    for (var i = 2; i <= num; i++) {
        if (num % i == 0 && i != num) {
            notPrime = true;
        }
    }
    if (notPrime == false) {
        console.log(num);
    }
}
```

Output:



```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\primenumber.js
0
1
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
PS F:\Pushkar\MCA\Sem 1\WAT>
```

Conclusion:

Implemented various concepts of REPL and performed various Node.js programs.

Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 2**	
Title of Lab Assignment: Create an application to demonstrate Node.js modules.			
DOP: 12-09-2023		**DOS: 13-09-2023**	
CO Mapped: **CO1**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No. 2

Aim:

Create an application to demonstrate the Node.js modules.

1) Built-in modules

1. Write a program to print information about the computer's operating system using the OS module (Use any 5 methods).
2. Print "Hello" every 500 milliseconds using the timer module. The message should be printed exactly 10 times. Use SetInterval, ClearInterval and SetTimeout methods.

2) Custom Modules

a) Create a Calculator Node.js Module with functions add, subtract and multiply, Divide. And use the Calculator module in another Node.js file.

b) Create a circle module with functions to find the area and perimeter of a circle and use it.

Theory:

1. **Built-in Module:**

   In Node.js, built-in modules are pre-existing libraries and modules that are included with the Node.js runtime environment. These modules provide a wide range of functionalities to help developers perform common tasks and interact with various aspects of the system. Below are some of the examples of built in modules.

   a) **fs (File System):** Used for reading and writing files, as well as manipulating directories.

   b) **http and https:** Modules for creating HTTP and HTTPS servers and making HTTP requests.

   c) **os (Operating System):** Provides information about the operating system.

   d) **path:** Helps in working with file and directory paths.

   e) **events:** Allows you to create and handle custom events.

   f) **crypto:** Offers cryptographic functionality for hashing, encryption, and decryption.

   g) **util:** Provides utility functions for debugging and formatting.

2. **Custom Module:**

   Custom modules in Node.js are user-defined JavaScript files that encapsulate specific functionality, making code more organized, modular, and reusable.

   a) Organize code into smaller, manageable units.

   b) Promote code reusability across your application.

   c) Improve code readability and maintainability.

   d) Extend Node.js capabilities beyond built-in modules.

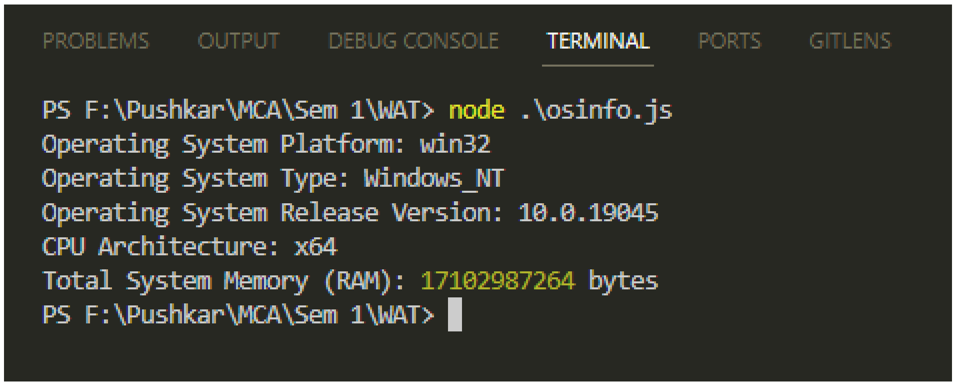
   Custom modules are essential for building scalable and maintainable Node.js applications by facilitating code modularization and promoting best practices in software development.

1) **Write a program to print information about the computer's operating system using the OS module (use any 5 methods).**

Code:

```
const os = require('os');
console.log('Operating System Platform:', os.platform());
console.log('Operating System Type:', os.type());
console.log('Operating System Release Version:', os.release());
console.log('CPU Architecture:', os.arch());
console.log('Total System Memory (RAM):', os.totalmem(), 'bytes');
```

Output:



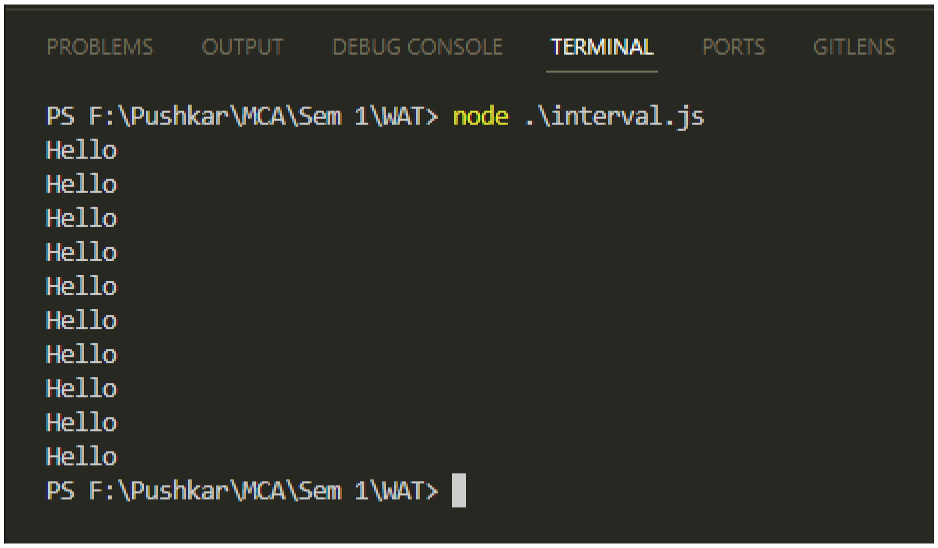
```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\osinfo.js
Operating System Platform: win32
Operating System Type: Windows_NT
Operating System Release Version: 10.0.19045
CPU Architecture: x64
Total System Memory (RAM): 17102987264 bytes
PS F:\Pushkar\MCA\Sem 1\WAT>
```

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 intervalId = setInterval(() => {
    console.log("Hello");
    count++;
    if(count == 10){
        clearInterval(intervalId);
    }
}, 500);
setTimeout(() => {
    clearInterval(intervalId);
},5500);
```

Output:



```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\interval.js
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
PS F:\Pushkar\MCA\Sem 1\WAT>
```

3) Create a Calculator Node.js Module with functions add, subtract and multiply, Divide. And use the Calculator module in another Node.js file.

<u>Code:</u>

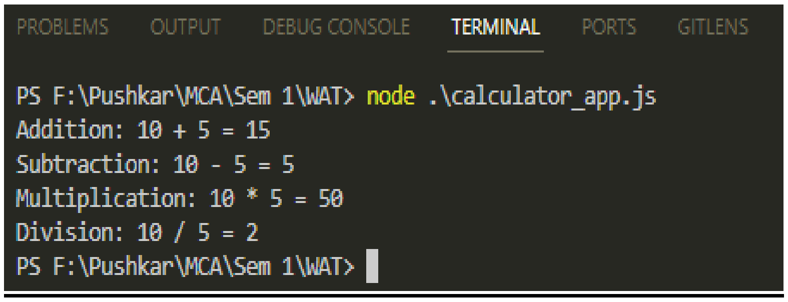
Calculator.js

```
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("Divide by zero error!")
    }
    return(a / b);
}

module.exports = {
    add,
    subtract,
    multiply,
    divide,
};
```

Calculator_App.js

```
const calculator = require('./calculator');
const num1 = 10;
const num2 = 5;
console.log(`Addition: ${num1} + ${num2} = ${calculator.add(num1, num2)}`);
console.log(`Subtraction: ${num1} - ${num2} = ${calculator.subtract(num1, num2)}`);
console.log(`Multiplication: ${num1} * ${num2} = ${calculator.multiply(num1, num2)}`);
console.log(`Division: ${num1} / ${num2} = ${calculator.divide(num1, num2)}`);
```

<u>Output:</u>



```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\calculator_app.js
Addition: 10 + 5 = 15
Subtraction: 10 - 5 = 5
Multiplication: 10 * 5 = 50
Division: 10 / 5 = 2
PS F:\Pushkar\MCA\Sem 1\WAT>
```

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

Code:

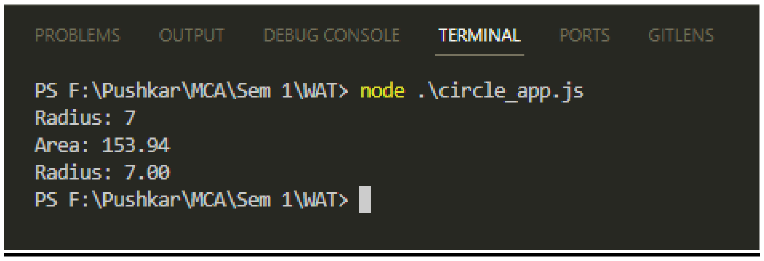
Circle.js

```
function CalculateArea(radius){
   return Math.PI * Math.pow(radius, 2);
}
function calculatePerimeter(radius){
   return 2 * Math.PI * radius;
}
module.exports = {
   CalculateArea,
   calculatePerimeter,
};
```

Circle_App.js

```
const circle = require('./circle');
const radius = 7;
const area = circle.CalculateArea(radius);
const perimeter = circle.calculatePerimeter(radius);
console.log(`Radius: ${radius}`);
console.log(`Area: ${area.toFixed(2)}`);
console.log(`Radius: ${radius.toFixed(2)}`);
```

Output:



```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\circle_app.js
Radius: 7
Area: 153.94
Radius: 7.00
PS F:\Pushkar\MCA\Sem 1\WAT>
```

Conclusion:

Created an application by using Node.js modules such as,

- Built-in Module
- Custom Module

Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 3**	
Title of Lab Assignment: Create an application to demonstrate Node.js Event Emitter.			
DOP: 25-09-2023		**DOS: 26-09-2023**	
CO Mapped: **CO1**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No. 3

Aim:

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

Description:

Understanding the Node.js EventEmitter Module

- **Purpose of EventEmitter:**
  The EventEmitter module is a fundamental component of Node.js, designed to facilitate the implementation of event-driven programming. Node.js is known for its asynchronous, non-blocking I/O model, and EventEmitter plays a pivotal role in building scalable and responsive applications by allowing different parts of the code to communicate efficiently through events.

- **Key Components:**
  1. EventEmitter Class:
     a. The core of the EventEmitter module is the EventEmitter class. It provides methods for emitting events and registering event listeners.
     b. Developers can create instances of this class to manage custom events within their applications.

  2. Event:
     a. An event is a signal or notification that something specific has happened. It is identified by a unique name or identifier.
     b. Events can be emitted (triggered) by an EventEmitter instance when a particular action or condition occurs.

  3. Event Listener:
     a. An event listener is a function that is registered to respond to a specific event when it is emitted.
     b. Event listeners are responsible for handling events by executing custom code in response to the event.

- **Example Usage:**

  Let's illustrate the usage of the EventEmitter module with a simple example:

  ```
  const EventEmitter = require('events');
  const myEmitter = new EventEmitter(); // Register an event listener
  myEmitter.on('greet', (name) => {
          console.log(`Hello, ${name}!`);
  }); // Emit the 'greet' event
  myEmitter.emit('greet', 'John');
  ```

  In this example, we:

  1. Import the `events` module and create an instance of `EventEmitter`.
  2. Register an event listener for the 'greet' event.
  3. Emit the 'greet' event with a 'name' argument, triggering the event listener to execute and print a greeting.
  4. This simple example demonstrates the EventEmitter's role in event-driven programming, where events are emitted and listeners respond to them.

- **Practical Applications**

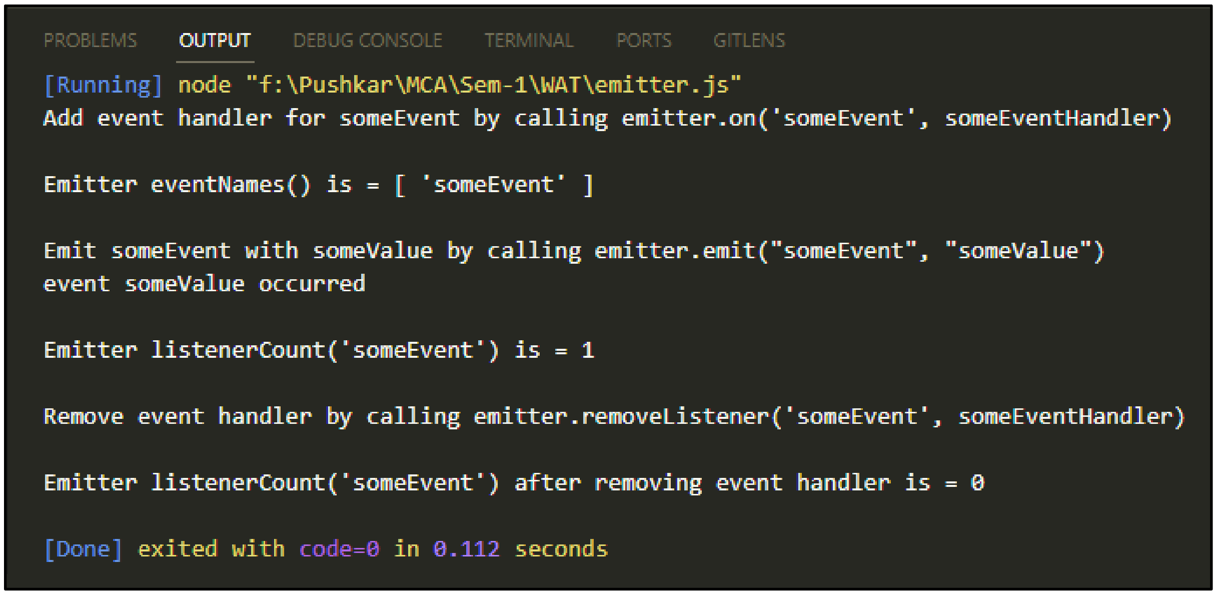
  1. HTTP Servers: EventEmitter is extensively used in building Node.js HTTP servers. HTTP requests, such as 'request' and 'response' events, are handled using EventEmitter to make servers highly responsive.
  2. File I/O: Reading and writing files asynchronously often involve EventEmitter. Events like 'data' and 'end' are emitted during file streams to handle data chunks and stream completion.
  3. Custom Applications: Developers can create custom events and listeners for various scenarios within their applications. For instance, handling user interactions, real-time updates, or custom triggers.

1. Create an application to demonstrate various Node.js Events in the event emitter class.

Code:

```
const events = require('events');
const someEventHandler = x => console.log('event', x, 'occurred');
const emitter = new events.EventEmitter();

console.log("Add event handler for someEvent by calling emitter.on('someEvent',
someEventHandler)");
emitter.on('someEvent', someEventHandler);

console.log("\nEmitter eventNames() is =", emitter.eventNames());
console.log('\nEmit someEvent with someValue by calling emitter.emit("someEvent",
"someValue")');

emitter.emit('someEvent', 'someValue');
console.log("\nEmitter listenerCount('someEvent') is =",
emitter.listenerCount('someEvent'));
console.log("\nRemove event handler by calling emitter.removeListener('someEvent',
someEventHandler)");
emitter.removeListener('someEvent', someEventHandler);
console.log("\nEmitter listenerCount('someEvent') after removing event handler is =",
emitter.listenerCount('someEvent'));
```

Output:



```
[Running] node "f:\Pushkar\MCA\Sem-1\WAT\emitter.js"
Add event handler for someEvent by calling emitter.on('someEvent', someEventHandler)

Emitter eventNames() is = [ 'someEvent' ]

Emit someEvent with someValue by calling emitter.emit("someEvent", "someValue")
event someValue occurred

Emitter listenerCount('someEvent') is = 1

Remove event handler by calling emitter.removeListener('someEvent', someEventHandler)

Emitter listenerCount('someEvent') after removing event handler is = 0

[Done] exited with code=0 in 0.112 seconds
```

2. Create functions to sort, reverse and search for an element in an array. Register and trigger these functions using events.

Code

array_operation.js

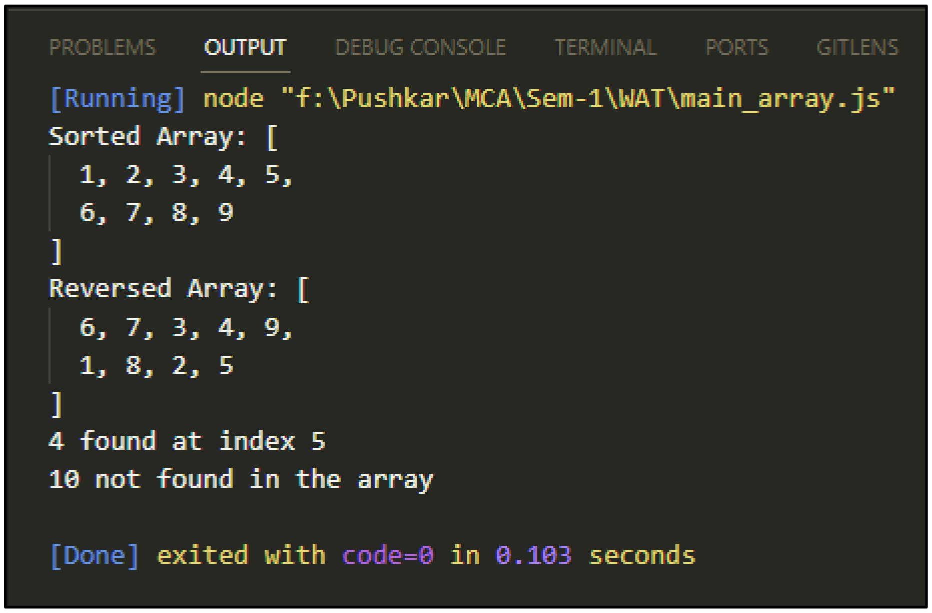
```
const EventEmitter = require('events');
class ArrayOperations extends EventEmitter {
  constructor() {
  super();
  }
  // Function to sort an array
  sortArray(arr) {
    const sortedArray = arr.slice().sort((a, b) => a - b);
    this.emit('sorted', sortedArray);
  }

  // Function to reverse an array
  reverseArray(arr) {
    const reversedArray = arr.slice().reverse();
    this.emit('reversed', reversedArray);
  }
  // Function to search for an element in an array
  searchArray(arr, target) {
    const index = arr.indexOf(target);
    this.emit('searched', { target, index });
  }
}
module.exports = ArrayOperations;
```

array_main.js

```
const ArrayOperations = require('./arrayOperations');
const arrayOps = new ArrayOperations();

// Register event listeners
arrayOps.on('sorted', (sortedArray) => {
  console.log('Sorted Array:', sortedArray);
});

arrayOps.on('reversed', (reversedArray) => {
  console.log('Reversed Array:', reversedArray);
});
```

```
arrayOps.on('searched', ({ target, index }) => {
    if (index !== -1) {
        console.log(`${target} found at index ${index}`);
    } else {
        console.log(`${target} not found in the array`);
    }
});

// Sample array
const myArray = [5, 2, 8, 1, 9, 4, 3, 7, 6];

// Trigger the functions using events
arrayOps.sortArray(myArray);
arrayOps.reverseArray(myArray);
arrayOps.searchArray(myArray, 4);
arrayOps.searchArray(myArray, 10);
```

Output:



```
[Running] node "f:\Pushkar\MCA\Sem-1\WAT\main_array.js"
Sorted Array: [
  1, 2, 3, 4, 5,
  6, 7, 8, 9
]
Reversed Array: [
  6, 7, 3, 4, 9,
  1, 8, 2, 5
]
4 found at index 5
10 not found in the array

[Done] exited with code=0 in 0.103 seconds
```

Conclusion:

In conclusion, the Node.js EventEmitter module is a cornerstone of Node.js development, enabling event-driven programming in both core modules and custom applications. It serves as a powerful communication mechanism, allowing different parts of an application to interact efficiently by emitting and responding to events.

The EventEmitter module is vital for building scalable and responsive applications that can handle numerous simultaneous events and asynchronous operations. Its use cases extend from HTTP servers and file I/O to custom application scenarios, making it a fundamental skill for Node.js developers.

By understanding and effectively utilizing EventEmitter, developers can harness the full potential of Node.js's event-driven architecture, resulting in robust, modular, and highly responsive applications that meet the demands of modern web and server development.

Name of Student: Pushkar Sane			
Roll Number: 45		Lab Assignment Number: 4	
Title of Lab Assignment: Create an application to demonstrate Node.js Functions-timer function.			
DOP: 26-09-2023		DOS: 28-09-2023	
CO Mapped: CO1	PO Mapped: PO3, PO5, PSO1, PSO2	Faculty Signature:	Marks:

Practical No. 4

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

Description:

- **Node.js Functions - Timer Function**

  In Node.js, timer functions are essential tools for scheduling and executing code at specified intervals. These timer functions are a fundamental part of Node.js's asynchronous and event-driven architecture, making them useful for various tasks, including periodic data fetching, task automation, real-time updates, and more. Among these timer functions, one of the most commonly used is the \`setInterval\` function for creating timer-based functionality.

- **Key Concepts**
  1. **\`setInterval\` Function:**
     a. The \`setInterval\` function is a built-in Node.js feature that allows you to repeatedly execute a specified function at a fixed time interval.
     b. It takes two arguments: the function to execute and the time interval in milliseconds.
     c. For example, to execute a function every 5 seconds, you can use \`setInterval(myFunction, 5000)\`.
  2. **\`setTimeout\` Function:**
     a. While \`setInterval\` repeatedly executes a function at intervals, the \`setTimeout\` function executes a function once after a specified delay in milliseconds.
     b. It is often used for tasks that require a one-time delay.
  3. **Use Cases:**
     a. Periodic Tasks: Timer functions are commonly used for tasks that need to occur at regular intervals, such as sending heartbeats, checking for updates, or refreshing data.

   b. Real-time Applications: In real-time applications like chats or online games, timer functions can be used to update and synchronize data among clients.
   c. Task Scheduling: Node.js timers are helpful for scheduling tasks, enabling automation of repetitive processes.

4. **Clearing Timers:**
   a. To stop a timer created with `setInterval` or `setTimeout`, you can use the `clearInterval` or `clearTimeout` functions, respectively.
   b. Clearing timers is important to prevent memory leaks and ensure efficient resource management in Node.js applications.

5. **Asynchronous Nature:**
   a. Timer functions in Node.js are non-blocking and asynchronous, allowing you to perform other tasks while waiting for the specified time interval to elapse.

6. **Edge Cases:**
   a. It's important to consider edge cases and potential issues when working with timers, such as timer drift (small variations in timing) and ensuring that the timer's function does not block the event loop.

Q.1. Create an application to demonstrate Node.js Functions-timer function(displays every 10 seconds).

Code:

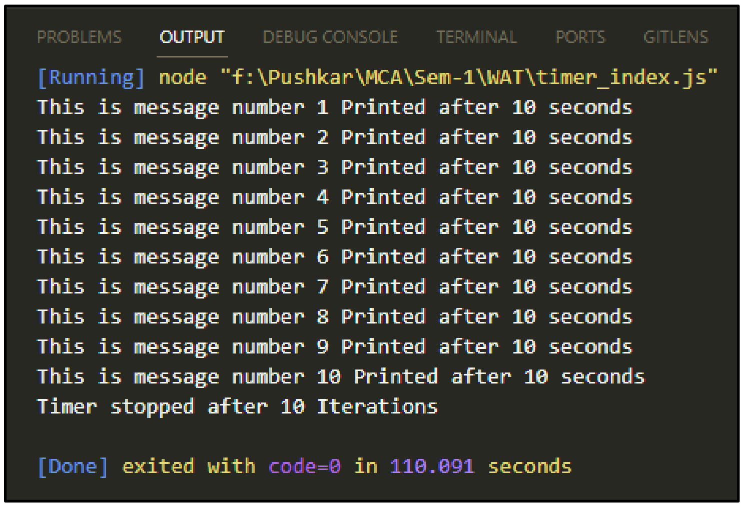
Timer.js

```
let count = 0;
let interval;
function timerFunction() {
   console.log("This is message number " + (count + 1) + " Printed after 10 seconds");
   count++;
}

module.exports = {
   startTimer: function() {
      interval = setInterval(timerFunction, 10000);
   },

   stopTimer: function() {
      clearInterval(interval);
      console.log("Timer stopped after 10 Iterations");
   }
};
```

Timer_index.js

```
const timer = require('./timer');
timer.startTimer();
// Using setTimeout to stop the timer after 10 iterations
setTimeout(timer.stopTimer, 110000);
```

Output:



```
[Running] node "f:\Pushkar\MCA\Sem-1\WAT\timer_index.js"
This is message number 1 Printed after 10 seconds
This is message number 2 Printed after 10 seconds
This is message number 3 Printed after 10 seconds
This is message number 4 Printed after 10 seconds
This is message number 5 Printed after 10 seconds
This is message number 6 Printed after 10 seconds
This is message number 7 Printed after 10 seconds
This is message number 8 Printed after 10 seconds
This is message number 9 Printed after 10 seconds
This is message number 10 Printed after 10 seconds
Timer stopped after 10 Iterations

[Done] exited with code=0 in 110.091 seconds
```

Conclusion:

Node.js timer functions, particularly `setInterval` and `setTimeout`, are powerful tools for adding time based functionality to your applications. They are instrumental in tasks that require periodic execution, synchronization, and automation. However, developers should be mindful of potential edge cases and handle timers carefully to ensure optimal performance and reliability in their Node.js applications. Properly managed timers can enhance the responsiveness and efficiency of your code, making them a fundamental aspect of Node.js development.

Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 4**	
Title of Lab Assignment: Using file handling to demonstrate all basic file operations (create, write, read, delete).			
DOP: 09-10-2023		**DOS: 10-10-2023**	
CO Mapped: **CO1**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No. 5

Aim:

Using file handling to demonstrate all basic file operations (Create, write, read, delete).

Theory:

File Handling in Node.js:

1. The most important functionalities provided by programming languages are Reading and Writing files from computers. Node.js provides the functionality to read and write files from the computer.
2. Reading and Writing the file in Node.js is done by using one of the coolest Node.js modules called fs module, it is one of the most well-known built-in Node.js modules out there.
3. The file can be read and written in node.js in both Synchronous and Asynchronous ways.
4. A Synchronous method is a code-blocking method which means the given method will block the execution of code until its execution is finished (i.e. Complete file is read or written).
5. On the other hand, an Asynchronous method has a callback function that is executed on completion of the execution of the given method and thus allows code to run during the completion of its execution.

Creating a file:

To create a new file in Node.js, you can use the `fs.writeFile()` method. This method allows you to specify the filename, the content to be written to the file, and a callback function to handle any potential errors. Here's an example:

Example:

```
const fs = require('fs');
fs.writeFile('sample.txt', 'This is a sample file created in Node.js.', (err) => {
  if (err) throw err;
  console.log('File created successfully!');
});
```

Writing to a File:

To add data to an existing file or append to its content, you can use the `fs.appendFile()` method. This method allows you to append data to a file without overwriting its existing content:

Example:

```
const fs = require('fs');
fs.appendFile('sample.txt', '\nThis is additional text.', (err) => {
  if (err) throw err;
  console.log('Data appended to the file!');
});
```

Reading from a File:

To read data from a file in Node.js, you can use the `fs.readFile()` method. This method reads the content of a file and provides it as a callback parameter for further processing:

Example:

```
const fs = require('fs');
fs.readFile('sample.txt', 'utf8', (err, data) => {
  if (err) throw err;
console.log('File content:\n', data);
});
```

Deleting a File:

To delete a file in Node.js, you can use the `fs.unlink()` method. This method takes the filename as an argument and deletes the file:

Example:

```
const fs = require('fs');
fs.unlink('sample.txt', (err) => {
  if (err) throw err;
  console.log('File deleted successfully!');
});
```

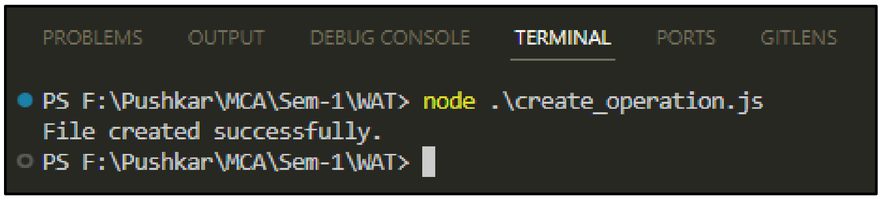
When working with file operations in Node.js, it's important to handle errors gracefully, as various issues like file permission problems or file not found errors can occur. The provided code examples include error handling to ensure robust file handling in your Node.js applications.

Code:

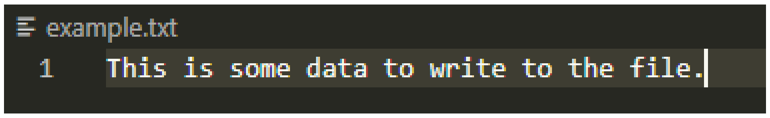
1. **Creating a File (Write Operation):**

```
const fs = require('fs');
// Data to be written to the file
const dataToWrite = 'This is some data to write to the file.';
// Specify the filename and content
fs.writeFile('example.txt', dataToWrite, (err) => {
  if (err) {
    console.error('Error writing to the file:', err);
  } else {
    console.log('Hello World in this file from Sweety ');
  }
});
```

Output:



```
PS F:\Pushkar\MCA\Sem-1\WAT> node .\create_operation.js
File created successfully.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

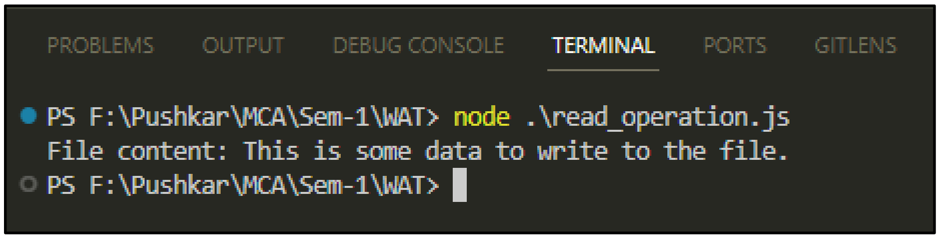


```
example.txt
This is some data to write to the file.
```

2. **Reading a File (Read Operation):**

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

Output:

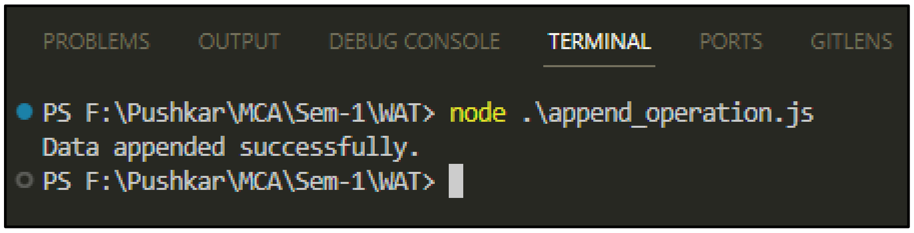


```
PS F:\Pushkar\MCA\Sem-1\WAT> node .\read_operation.js
File content: This is some data to write to the file.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

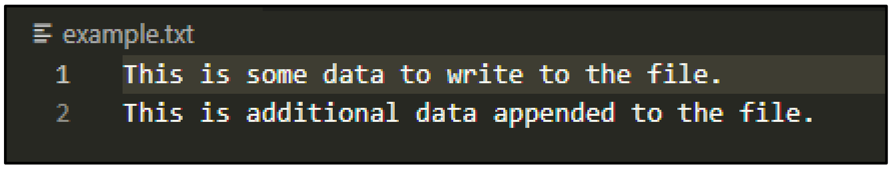
3. **Appending to a File (Write Operation):**

```
const fs = require("fs");
const newDataToAppend = "\nThis is additional data appended to the file.";
fs.appendFile("example.txt", newDataToAppend, (err) => {
  if (err) {
    console.error("Error appending to the file:", err);
  } else {
    console.log("Data appended successfully.");
  }
});
```

Output:



```
PS F:\Pushkar\MCA\Sem-1\WAT> node .\append_operation.js
Data appended successfully.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

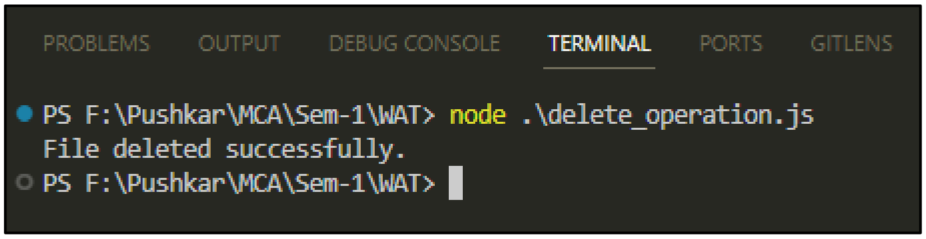


```
This is some data to write to the file.
This is additional data appended to the file.
```

4. **Deleting a File (Delete Operation):**

```
const fs = require("fs");
fs.unlink("example.txt", (err) => {
  if (err) {
    console.error("Error deleting the file:", err);
  } else {
    console.log("File deleted successfully.");
  }
});
```

Output:



```
PS F:\Pushkar\MCA\Sem-1\WAT> node .\delete_operation.js
File deleted successfully.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

Conclusion:

By showcasing basic file operations in Node.js, including file creation, writing, reading, and deletion, we've illustrated the fundamental capabilities of file handling in a programming environment. These operations form the building blocks for managing data persistence and interaction with the file system within Node.js applications. Mastery of these operations is essential for effective file management and data manipulation in various software solutions.

Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 6**	
Title of Lab Assignment: **1) Create a HTTP Server and serve HTML, CSV, JSON and PDF Files.** **2) Create a HTTP Server and stream a video file using piping.** **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.**			
DOP: 11-10-2023		**DOS: 13-10-2023**	
CO Mapped: **CO1**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No. 6

Aim:

1. Create a HTTP Server and serve HTML,CSV,JSON and PDF Files.
2. Create a HTTP Server and stream a video file using piping.
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.

Description:

1. **Create an HTTP Server and Serve Various File Types (HTML, CSV, JSON, PDF)**
   - **HTTP Server:**
     An HTTP (Hypertext Transfer Protocol) server is a software application that handles incoming requests from clients, such as web browsers, and responds with the appropriate content. It listens on a specific port (commonly port 80 for HTTP) and communicates using the HTTP protocol.
   - **Content Types and `Content-Type` Header:**
     The `Content-Type` header in an HTTP response specifies the type of data being sent. It helps the client (browser) understand how to interpret the content. For example, HTML files have a `Content-Type` of 'text/html', CSV files are 'text/csv', JSON files are 'application/json', and PDF files are 'application/pdf'.
   - **Serving Files:**
     To serve files, the server reads the file's content and sends it as the response. The `ContentType` header is set according to the file's type, and the browser renders the content appropriately.
   - **File Streaming:**
     When serving large files, like PDFs, it's more efficient to use streaming. Instead of loading the entire file into memory before sending it, you send chunks of data as they are read. This is memory-efficient and allows users to start viewing or downloading the content more quickly.

- **HTTP Server:**
  Component: The Node.js `http` module is used to create an HTTP server. It listens for incoming HTTP requests and manages the communication.

- **File System Interaction:**
  Component: The `fs` (File System) module in Node.js is used to read files from the server's file system.

- **Content-Type Header:**
  Component: The `Content-Type` header in the HTTP response is set to specify the type of the content being served.

- **Routing:**
  Component: In this basic example, routing is done using conditional logic based on the URL path requested.

- **File Streaming:**
  Component: In the case of large files like PDFs, streaming is achieved using Node.js's stream APIs. This efficiently sends file data in chunks from the server to the client.

2. **Create an HTTP Server and Stream a Video File Using Piping**
   - **HTTP Streaming:**
     HTTP streaming allows media content to be sent over an HTTP connection progressively. It's commonly used for videos and audio files. The server sends the content as it's available, and the client starts rendering or playing it without waiting for the entire file to download.

   - **Content-Disposition Header:**
     The `Content-Disposition` header can be used to suggest how the browser should handle the file. Setting it to 'inline' suggests that the file should be displayed in the browser, while 'attachment' suggests that the file should be downloaded.

- **Piping:**
  In Node.js, piping is a technique to efficiently transfer data from a source to a destination. It's especially useful when dealing with large files. A readable stream (source) is piped into a writable stream (destination), and data flows between them in chunks. This is memory-efficient and avoids loading the entire file into memory.

- **HTTP Streaming:**
  Component: HTTP streaming is the main concept, where data is sent over an HTTP connection progressively.

- **Content-Disposition Header:**
  Component: The `Content-Disposition` header can be used to suggest how the browser should handle the file. In this scenario, it might be set to 'inline' to prompt in-browser rendering.

- **Piping:**
  Component: The `pipe` method from Node.js stream APIs is used to efficiently transfer data from a source (file) to a destination (response). This technique is memory-efficient and suitable for streaming large files.

3. **Develop 3 HTML Web Pages and Serve Them Using Routing:**
   - **HTML Pages:**
     HTML (Hypertext Markup Language) is the standard language for creating web pages. It provides the structure and content of web documents. Browsers interpret HTML to render web pages.

   - **Routing:**
     Routing in web development involves mapping URLs to specific actions or responses on the server. It's essential for handling multiple pages or different functionalities within a web application. Routes determine which code should execute based on the URL requested by the client.

- **Server-Side Rendering:**
  Server-side rendering (SSR) is a technique where web pages are generated dynamically on the server and sent as fully rendered HTML to the client. This is in contrast to client-side rendering (CSR), where most rendering occurs in the browser using JavaScript. SSR is used for SEO optimization and better performance in some cases.

- **Express.js:**
  Express.js is a popular Node.js web application framework that simplifies building web applications and APIs. It provides tools for routing, middleware, and serving static files. In the example, Express is used to define routes and serve HTML pages based on the URL requested.

- **Express.js:**
  Component: Express.js is a Node.js web framework used to simplify routing and handling HTTP requests. It provides routing, middleware, and various tools for building web applications.

- **Routing:**
  Component: Express.js is used to define routes, which map URLs to specific functions or templates (HTML pages) to be served in response to different URL paths.

- **HTML Pages:**
  Component: HTML pages are the actual content that is served to the client. These pages are stored in the 'public' directory and sent to the client based on the requested route.

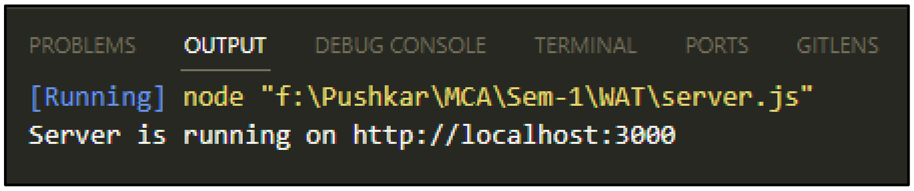
- **Server-Side Rendering:**
  Component: In the Express.js-based server, server-side rendering is the primary technique. It involves dynamically generating web pages on the server and sending fully rendered HTML pages to the client.

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

server.js

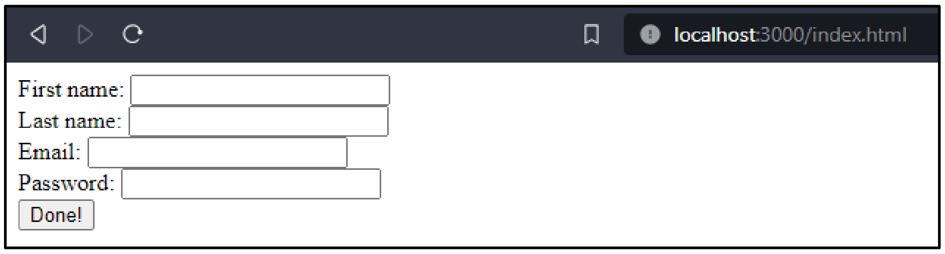
```
const http = require('http');
const fs = require('fs');
const path = require('path');
const server = http.createServer((req, res) => {
    const { url } = req;
    const filePath = path.join(__dirname, 'public', url);
    fs.readFile(filePath, (err, data) => {
        if (err) {
            res.writeHead(404, {'Content-Type': 'text/plain'});
            res.end('File not found');
        } else {
            let contentType = 'text/plain';
            if (filePath.endsWith('.html')) {
                contentType = 'text/html';
            } else if (filePath.endsWith('.csv')) {
                contentType = 'text/csv';
            } else if (filePath.endsWith('.json')) {
                contentType = 'application/json';
            } else if (filePath.endsWith('.pdf')) {
                contentType = 'application/pdf';
            }
            res.writeHead(200, {'Content-Type': contentType });
            res.end(data);
        }
    });
});
const port = 3000;
server.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});
```

Output:



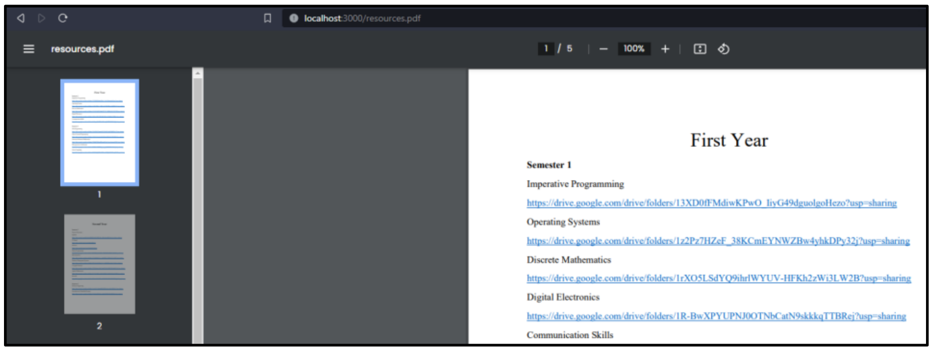
```
[Running] node "f:\Pushkar\MCA\Sem-1\WAT\server.js"
Server is running on http://localhost:3000
```

index.html (Serving html file)



First name:
Last name:
Email:
Password:

resources.pdf (Serving pdf file)



First Year

Semester 1

Imperative Programming

https://drive.google.com/drive/folders/13XD0fFMdiwKPwO_IiyG49dguolgoHezo?usp=sharing

Operating Systems

https://drive.google.com/drive/folders/1z2Pz7HZeF_38KCmEYNWZBw4yhkDPy32j?usp=sharing

Discrete Mathematics

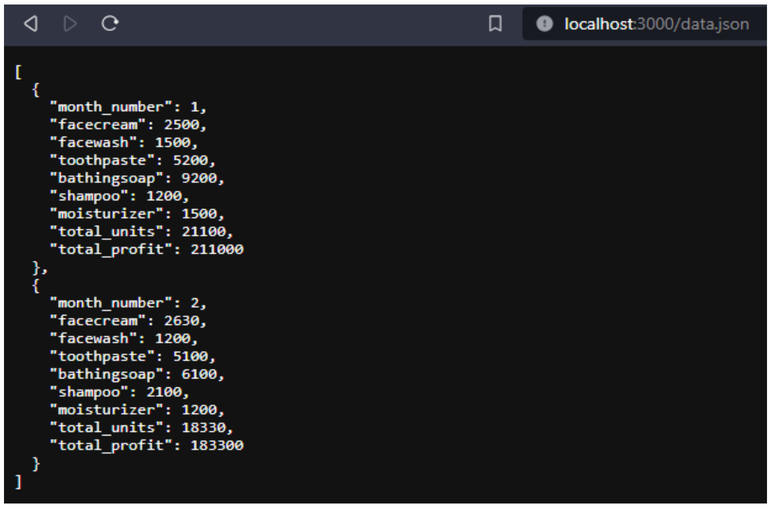
https://drive.google.com/drive/folders/1rXO5LSdYQ9ihrlWYUV-HFKh2zWi3LW2B?usp=sharing

Digital Electronics

https://drive.google.com/drive/folders/1R-BwXPYUPNJ0OTNbCatN9skkkqTTBRej?usp=sharing

Communication Skills

data.json (Serving as json file)



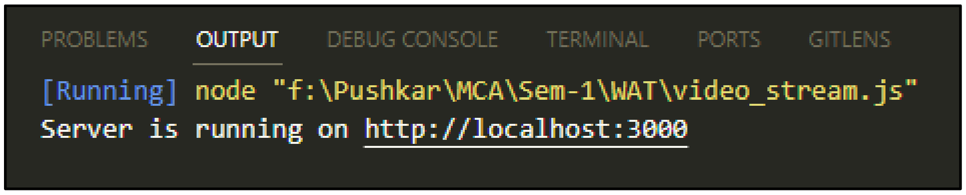
```
[
  {
    "month_number": 1,
    "facecream": 2500,
    "facewash": 1500,
    "toothpaste": 5200,
    "bathingsoap": 9200,
    "shampoo": 1200,
    "moisturizer": 1500,
    "total_units": 21100,
    "total_profit": 211000
  },
  {
    "month_number": 2,
    "facecream": 2630,
    "facewash": 1200,
    "toothpaste": 5100,
    "bathingsoap": 6100,
    "shampoo": 2100,
    "moisturizer": 1200,
    "total_units": 18330,
    "total_profit": 183300
  }
]
```

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

videosteam.js

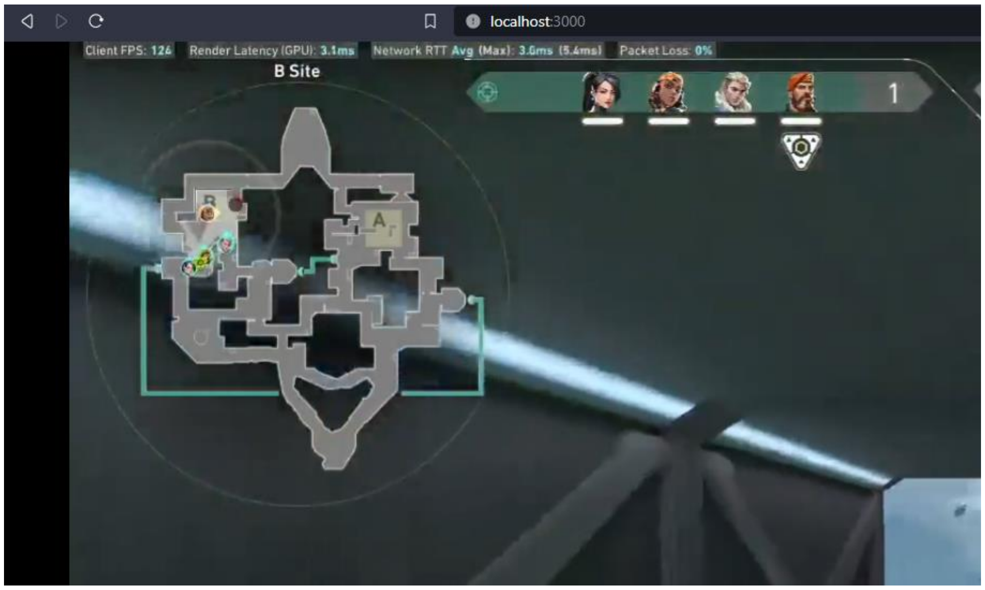
```
const http = require('http');
const fs = require('fs');
const path = require('path');
const server = http.createServer((req, res) => {
   const videoPath = path.join(__dirname, 'public', 'video.mp4');
   const stat = fs.statSync(videoPath);
   res.writeHead(200, {
      'Content-Type': 'video/mp4',
      'Content-Length': stat.size,
   });
   const videoStream = fs.createReadStream(videoPath);
   videoStream.pipe(res);
});
const port = 3000;
server.listen(port, () => {
   console.log(`Server is running on http://localhost:${port}`);
});
```

Output:



```
[Running] node "f:\Pushkar\MCA\Sem-1\WAT\video_stream.js"
Server is running on http://localhost:3000
```

video.mp4 (Streaming video using piping).


localhost:3000
Client FPS: 126
Render Latency (GPU): 3.1ms
Network RTT Avg (Max): 3.0ms (5.4ms)
Packet Loss: 0%
B Site
1

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.**

index.js

```
<!DOCTYPE html>
<html>
   <head>
      <title>College Home Page</title>
   </head>
   <body>
      <h1>Welcome to Our College</h1>
      <p>We are a leading institution dedicated to providing high-quality education.</p>
   </body>
   <p><a href="about.html">About us</a></p>
   <p><a href="contact.html">Contact us</a></p>
   <p><a href="index.html">Home</a></p>
</html>
```

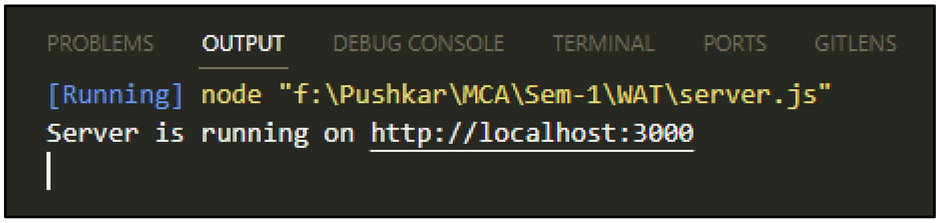
about.js

```
<!DOCTYPE html>
<html>
  <head>
    <title>About Me</title>
  </head>
  <body>
    <h1>About Me</h1>
    <p>I am a student at this college, and I love programming and web development.</p>
  </body>
  <p><a href="index.html">Home</a></p>
</html>
```

contact.js

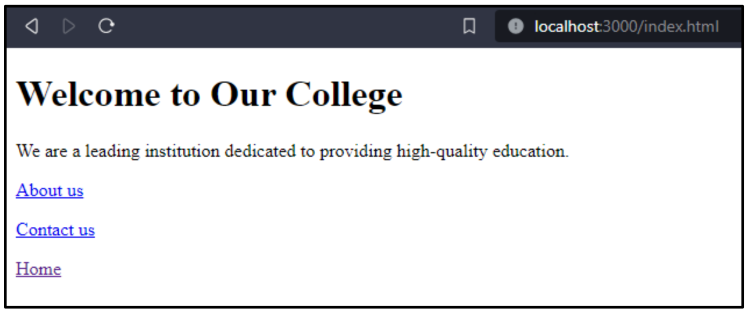
```
<!DOCTYPE html>
<html>
  <head>
    <title>Contact</title>
  </head>
  <body>
    <h1>Contact Us</h1>
    <p>Feel free to get in touch with us by email or phone.</p>
  </body>
  <p><a href="index.html">Home</a></p>
</html>
```

Output:



```
[Running] node "f:\Pushkar\MCA\Sem-1\WAT\server.js"
Server is running on http://localhost:3000
```

Index.html (Rendering Page 1)



Welcome to Our College

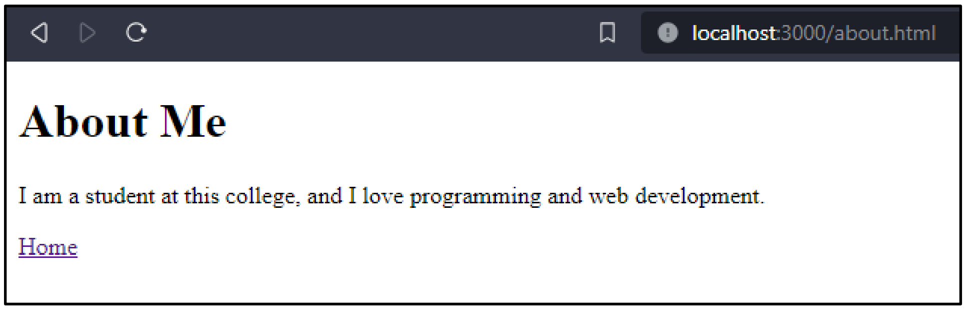
We are a leading institution dedicated to providing high-quality education.

About us

Contact us

Home

About.html (Rendering Page 2)

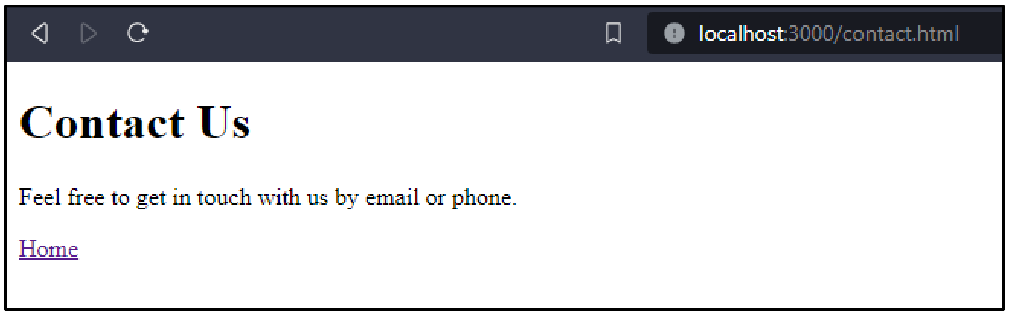


About Me

I am a student at this college, and I love programming and web development.

Home

Contact.html (Rendering Page 3)



Contact Us

Feel free to get in touch with us by email or phone.

Home

Conclusion:

We learned how to handle various file types (HTML, CSV, JSON, PDF) and ensure that the appropriate Content-Type and Content-Disposition headers are set, allowing for both file downloads and in-browser content rendering, depending on the file type.

Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 7**	
Title of Lab Assignment: Create an application to establish a connection with the MySQL database and perform basic database operations on it (student db consisting roll no, name, address), insert 10 records, update a particular student's record, delete a record.			
DOP: 16-10-2023		**DOS: 17-10-2023**	
CO Mapped: **CO2**	**PO Mapped:** **PO5, PSO1**	**Faculty Signature:**	**Marks:**

Practical No. 7

Aim: Create an application to establish a connection with the MySQL database and perform basic database operations on it (student db consisting roll no, name, address), insert 10 records, update a particular student's record, delete a record.

Description:

- **MySQL Server:**
    - MySQL is an open-source relational database management system used to store and manage data in a structured manner.

- **Node.js MySQL Driver:**
    - To connect to MySQL from Node.js, you'll need a Node.js MySQL driver. The `mysql` package is a commonly used choice for this purpose. You can install it using `npm`.
      ```bash
      npm install mysql
      ```
- **Code File (e.g. `app.js`):**
    - You'll write your Node.js code in a JavaScript file. This file will contain the logic for connecting to the database, executing queries, and handling the results.

- **Connection Configuration:**
    - Configure your MySQL connection by specifying the following details:
    - Host: The MySQL server's hostname or IP address.
    - User: The MySQL username with appropriate privileges.
    - Password: The password for the MySQL user.
    - Database: The name of the database you want to connect to.

- **Creating a Connection:**
    - Use the MySQL driver to create a connection to the MySQL database. This connection object will be used to perform database operations
      ```javascript
      const mysql = require('mysql');
      const connection = mysql.createConnection({
      ```

```
host: 'localhost',
user: 'your_mysql_username',
password: 'your_mysql_password',
database: 'your_database_name',
});
```

- **Handling Connection Events:**
    - You should handle events related to the database connection, such as errors and successful connections.

```javascript
connection.connect((err) => {
if (err) {
console.error('Error connecting to MySQL: ' + err.stack);
return;
}
console.log('Connected to MySQL as id ' + connection.threadId);
});
```

- **Performing Database Operations:**
    - You can use the `connection` object to execute SQL queries for various database operations, such as inserting, updating, and deleting records, as well as retrieving data (SELECT).

- **Error Handling:**
    - Implement error handling to gracefully deal with any issues that may arise during database operations or the connection process.

- **Closing the Connection:**
    - After you've completed your database operations, make sure to close the connection to free up resources and maintain security.

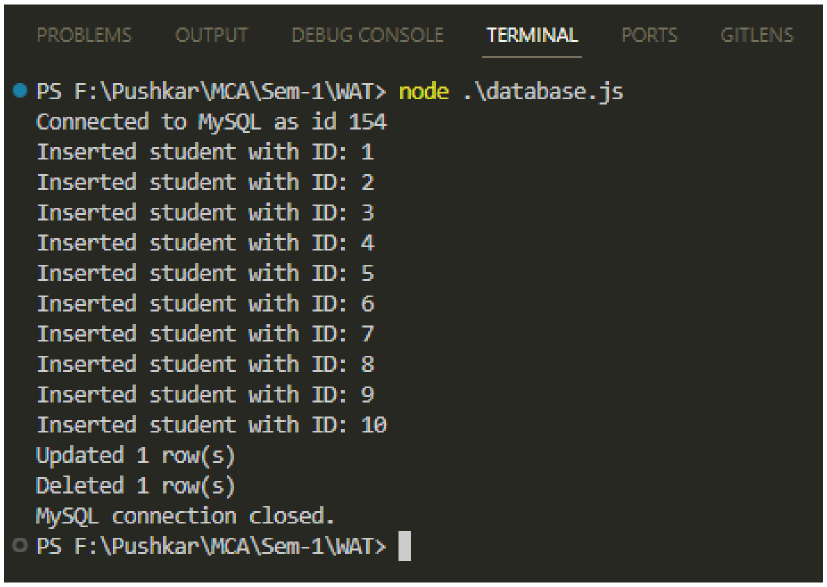
```javascript
connection.end((err) => {
if (err) {
console.error('Error closing the connection: ' + err.stack);
return;
}
console.log('MySQL connection closed.');
});
```
```

- We can create a Node.js application that connects to a MySQL database seamlessly. This setup is essential for building web applications, APIs, and other software that require database interaction.

**Code:**

**Database.js**

```
const mysql = require('mysql2');
// Create a connection to the MySQL database
const connection = mysql.createConnection({
   host: 'localhost',
   user: 'root',
   password: 'root123',
   database: 'student',
   connectionLimit : 10
});
// Connect to the MySQL server
connection.connect((err) => {
   if (err) {
      console.error('Error connecting to MySQL: ' + err.stack);
      return;
   } else {
      console.log('Connected to MySQL as id ' + connection.threadId);
   }
});
for (let i = 1; i <= 10; i++) {
   const student = {
      rollno: i,
      name: `student ${i}`,
      address: `Address ${i}`,
   };
```
```

```
connection.query('INSERT INTO students SET ?', student, (error, results) => {
  if (error) throw error;
  console.log(`Inserted student with ID: ${results.insertId}`);
  });
}
// Update a particular student's record
const updatedStudent = {
  name: 'Updated Student',
  address: 'Updated Address',
};
connection.query(
  'UPDATE students SET ? WHERE rollno = ?',
  [updatedStudent, 1],
  (error, results) => {
    if (error) throw error;
    console.log(`Updated ${results.affectedRows} row(s)`);
  }
);
// Delete a record (student with rollno 2)
connection.query('DELETE FROM students WHERE rollno = ?', 2, (error, results) => {
if (error) throw error;
  console.log(`Deleted ${results.affectedRows} row(s)`);
});
// Close the connection when done
connection.end((err) => {
  if (err) {
    console.error('Error closing the connection: ' + err.stack);
    return;
  }
console.log('MySQL connection closed.');
});
```

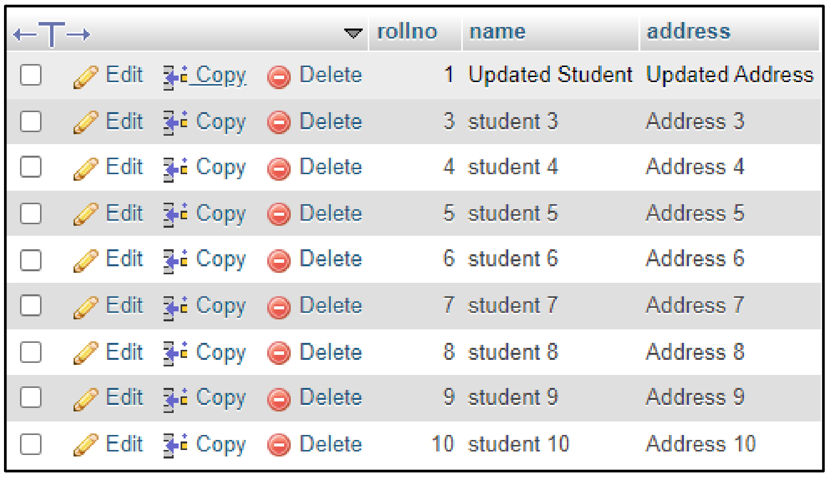
MySQL Queries

```
CREATE DATABASE student;
USE student;
CREATE TABLE students (
    rollno INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255)
);
select * from students;
```

Output:



```
PS F:\Pushkar\MCA\Sem-1\WAT> node .\database.js
Connected to MySQL as id 154
Inserted student with ID: 1
Inserted student with ID: 2
Inserted student with ID: 3
Inserted student with ID: 4
Inserted student with ID: 5
Inserted student with ID: 6
Inserted student with ID: 7
Inserted student with ID: 8
Inserted student with ID: 9
Inserted student with ID: 10
Updated 1 row(s)
Deleted 1 row(s)
MySQL connection closed.
PS F:\Pushkar\MCA\Sem-1\WAT>
```



	rollno	name	address
Edit Copy Delete	1	Updated Student	Updated Address
Edit Copy Delete	3	student 3	Address 3
Edit Copy Delete	4	student 4	Address 4
Edit Copy Delete	5	student 5	Address 5
Edit Copy Delete	6	student 6	Address 6
Edit Copy Delete	7	student 7	Address 7
Edit Copy Delete	8	student 8	Address 8
Edit Copy Delete	9	student 9	Address 9
Edit Copy Delete	10	student 10	Address 10

Conclusion:

Connecting MySQL to Node.js using Visual Studio Code (VSCode) is essential for building database-driven applications. By configuring the connection, handling events, and performing database operations, you can create robust, data-driven software efficiently and securely.

Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 8**	
Title of Lab Assignment: TypeScript installation, Environment Setup, Programs on decision making / functions / class & object.			
DOP: 16-10-2023		**DOS: 20-10-2023**	
CO Mapped: **CO4**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No. 8

Aim: TypeScript installation, Environment Setup, Programs on decision making/functions/class & object.

Description:

TypeScript Installation, Environment Setup, and How TypeScript Works:

TypeScript is a statically typed superset of JavaScript that provides optional static typing, enhanced tooling, and other features to improve the development experience. Here's a comprehensive note on installing TypeScript, setting up your development environment, and understanding how TypeScript works.

1. **TypeScript Installation:**
   To install TypeScript, you need to have Node.js and npm (Node Package Manager) on your system.
   Follow these steps:
   a. Install Node.js: Download and install Node.js from the [official Node.js website](https://nodejs.org/).
      Node.js comes with npm, which is required to install TypeScript.
   b. Install TypeScript:
      Open your command-line interface (CLI) and run the following command to install TypeScript globally on your system:
      bash
      npm install -g typescript
      This command installs TypeScript globally, making it available for use on your system.

2. **Environment Setup:**
   After installing TypeScript, you can set up your development environment to start writing TypeScript code.
   a. Create a Project Directory: Choose or create a directory where you want to work on your TypeScript project. You can do this through your file explorer or with the following command.

bash
mkdir my-typescript-project
cd my-typescript-project

b. Initialize a TypeScript Project: To configure your TypeScript project, create a `tsconfig.json` configuration file using the following command.
bash
tsc --init
This command generates a basic `tsconfig.json` file in your project directory. You can modify this file to customize TypeScript settings for your project.

c. Choose a Code Editor or IDE: Select a code editor or integrated development environment (IDE) that supports TypeScript. Popular choices include Visual Studio Code, WebStorm, and Sublime Text. Download and install your preferred editor.

Here's an overview of how TypeScript works and its key features:

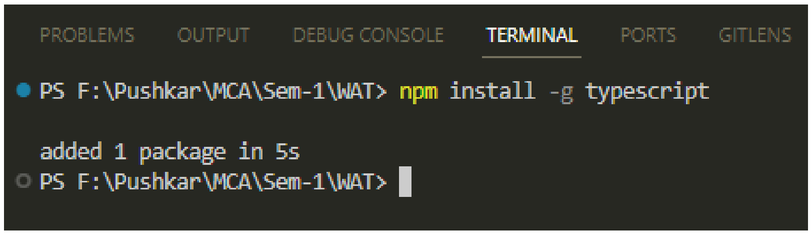
1. **TypeScript Code:**
   a. You write TypeScript code in `.ts` files. TypeScript supports modern JavaScript syntax, including ES6 and ESNext features.
   b. You can define variable types and use type annotations to provide information about the data types of variables, parameters, and function return values.
2. **Compilation:**
   a. TypeScript code is not directly understood by browsers or Node.js. To run TypeScript code, it must be transpiled to plain JavaScript.
   b. The TypeScript compiler, `tsc`, is used to transpile `.ts` files into `.js` files.
   c. The transpilation process involves checking for type errors, type inference, and removing TypeScriptspecific syntax. The output is JavaScript code that can be executed in a JavaScript runtime environment.
3. **Type Checking:**
   a. TypeScript provides optional static typing, allowing you to specify data types for variables and function parameters.
   b. The TypeScript compiler performs type checking during compilation. It flags type-related errors and provides warnings, helping to catch potential issues early in the development process.

4. **Tooling Support:**
    a. TypeScript offers enhanced tooling support, including autocompletion, refactoring, and navigation features in code editors and IDEs.
    b. Development environments such as Visual Studio Code provide built-in TypeScript support, making it easier to write, debug, and manage TypeScript projects.
5. **Execution:**
    a. The transpiled JavaScript code can be executed in any JavaScript environment, whether it's a web browser or Node.js.
    b. The runtime behavior of the code is the same as if it were originally written in JavaScript.

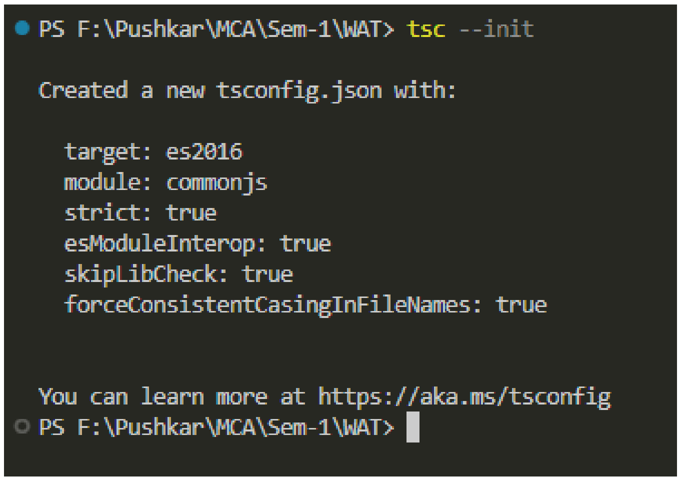
In conclusion, TypeScript is a valuable tool that enhances JavaScript development by providing static typing, enhanced tooling, and type checking while maintaining compatibility with JavaScript. By following the installation and environment setup steps, you can effectively work with TypeScript in your projects, catching errors early and improving code quality.

<u>Code:</u>

TypeScript installation, Environment Setup:



```
PS F:\Pushkar\MCA\Sem-1\WAT> npm install -g typescript

added 1 package in 5s
PS F:\Pushkar\MCA\Sem-1\WAT>
```



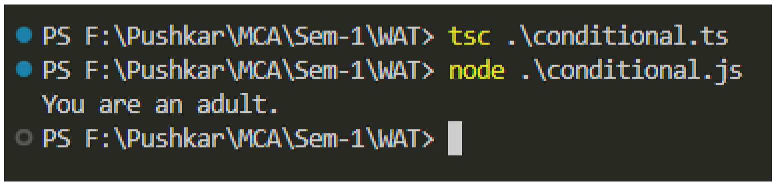
```
PS F:\Pushkar\MCA\Sem-1\WAT> tsc --init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true


You can learn more at https://aka.ms/tsconfig
PS F:\Pushkar\MCA\Sem-1\WAT>
```

1. **Program on decision making**

   a. **conditional.ts**

```
let age: number = 18;
if (age >= 18) {
   console.log("You are an adult.");
} else {
   console.log("You are a minor.");
}
```

Output:

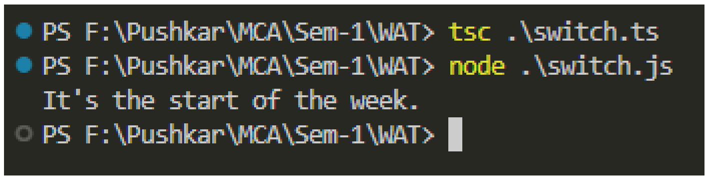


```
PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\conditional.ts
PS F:\Pushkar\MCA\Sem-1\WAT> node .\conditional.js
You are an adult.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

   b. **switch.ts**

```
let day: string = "Monday";
switch (day) {
   case "Monday":
      console.log("It's the start of the week.");
      break;
   case "Friday":
      console.log("It's almost the weekend!");
      break;
   default:
      console.log("It's a regular day.");
}
```

Output:



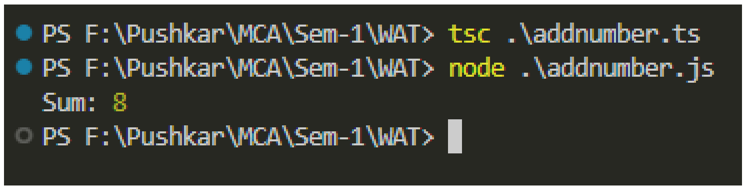
```
PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\switch.ts
PS F:\Pushkar\MCA\Sem-1\WAT> node .\switch.js
It's the start of the week.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

2. **Function Programs:**

   a. **addnumbers.ts**

```
function add(a: number, b: number): number {
    return a + b;
}
let result: number = add(5, 3);
console.log("Sum:", result);
```

Output:

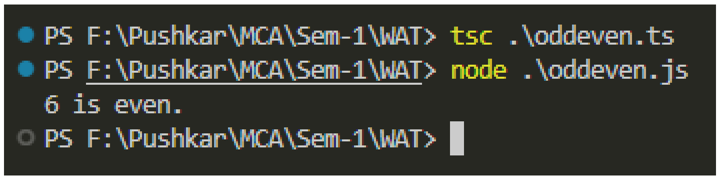


```
● PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\addnumber.ts
● PS F:\Pushkar\MCA\Sem-1\WAT> node .\addnumber.js
  Sum: 8
○ PS F:\Pushkar\MCA\Sem-1\WAT>
```

   b. **oddeven.ts**

```
function isEven(num: number): boolean {
    return num % 2 === 0;
}
let numberToCheck: number = 6;
if (isEven(numberToCheck)) {
    console.log(numberToCheck + " is even.");
} else {
    console.log(numberToCheck + " is odd.");
}
```

Output:



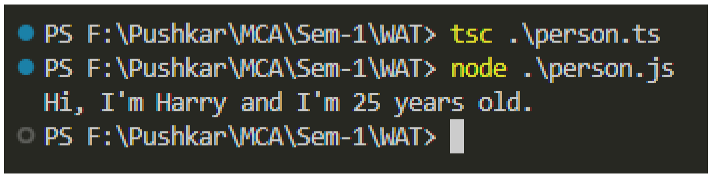
```
● PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\oddeven.ts
● PS F:\Pushkar\MCA\Sem-1\WAT> node .\oddeven.js
  6 is even.
○ PS F:\Pushkar\MCA\Sem-1\WAT>
```

3. Class and Object programs:

a. person.js

```
class Person {
  name: string;
  age: number;
  constructor(name: string, age: number) {
    this.name = name;
    this.age = age;
  }
  introduce(): string {
    return `Hi, I'm ${this.name} and I'm ${this.age} years old.`;
  }
}
let person1: Person = new Person("Alice", 30);
console.log(person1.introduce());
```

Output:



```
PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\person.ts
PS F:\Pushkar\MCA\Sem-1\WAT> node .\person.js
Hi, I'm Harry and I'm 25 years old.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

Conclusion:

By setting up TypeScript and leveraging its features, you can write more robust and maintainable JavaScript code while benefiting from a richer development experience. TypeScript's ability to catch errors early and enhance tooling support makes it a valuable addition to your development toolkit.

Name of Student: Pushkar Sane			
Roll Number: 45		**Practical Number: 9**	
Title of Lab Assignment: Introduction to Angular Setup for local development environment. Angular Architecture, create an application to demonstrate directives and pipes.			
DOP: 22-09-2023		**DOS: 23-09-2023**	
CO Mapped: **CO5**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No. 9

Aim: Introduction to Angular Setup for local development environment. Angular Architecture, Create an application to demonstrate directives and pipes.

Description:

Angular is a popular open-source web application framework developed by Google and a community of individual developers. It is designed to simplify the process of building dynamic, single-page web applications. To set up an Angular development environment, you'll need to follow several steps:

Prerequisites:

Before you start with Angular, ensure that you have the following prerequisites installed on your system:

1. **Node.js:**
   Angular uses Node.js for its build tools and package management. You can download and install it from Node.js website.
2. **npm (Node Package Manager):**
   npm comes bundled with Node.js. It is used to install Angular CLI and other dependencies.
3. **Angular CLI (Command Line Interface):**
   Angular CLI is a powerful command-line tool that makes it easy to create, build, test, and deploy Angular applications. To install Angular CLI, open your terminal or command prompt and run the following command:
   npm install -g @angular/cli
   This will globally install the Angular CLI tool.

Create a New Angular Project:

After Angular CLI is installed, you can create a new Angular project by running the following command:

ng new your-project-name

Replace your-project-name with your desired project name. Angular CLI will prompt you to configure various project settings, including whether to include Angular routing and which stylesheets (CSS, SCSS, etc.) to use.

Run the Development Server:

Navigate into your project folder:

cd your-project-name

Start the development server:

ng serve

This command will compile your Angular application and serve it on a development server. By default, you can access your application at http://localhost:4200/.

Project Structure:

Angular projects have a specific directory structure:

1. src/: This is where your application's source code resides.
2. app/: This folder contains the main application code.
3. index.html: The main HTML file for your application.
4. angular.json: Configuration file for Angular CLI.
5. package.json: Lists project dependencies and scripts.
6. tsconfig.json: Configuration for TypeScript.

Creating Components:

In Angular, your application is built using components. You can generate a new component using Angular CLI:

ng generate component component-name

Building and Deploying:

When you're ready to deploy your Angular application, you can use Angular CLI to build it:

ng build --prod

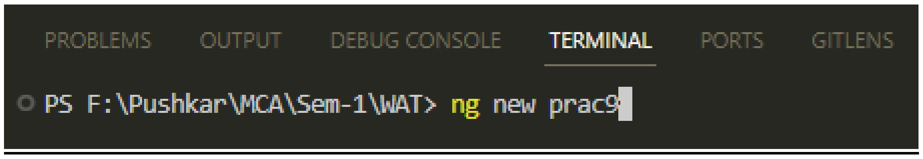
This will create a dist/ directory with optimized production-ready code. You can then host this folder on a web server.

This is a basic introduction to setting up an Angular development environment. Angular offers a rich ecosystem for building complex web applications with features like dependency injection, routing, and powerful templating. As you become more familiar with Angular, you can explore its documentation and community resources to build robust web applications.

1. Create an application to demonstrate directives and pipes

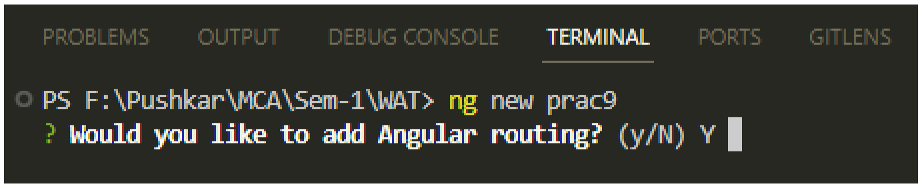
Steps to execute the practical:

1. Create a folder "practical 9"
2. Open the terminal (please use cmd, avoid using powershell) and navigate to "practical 9" folder by using the "cd" command
3. Install Angular CLI globally by running the following command
   npm install -g @angular/cli
4. Create a new angular project by running the following command
   ng new prac9



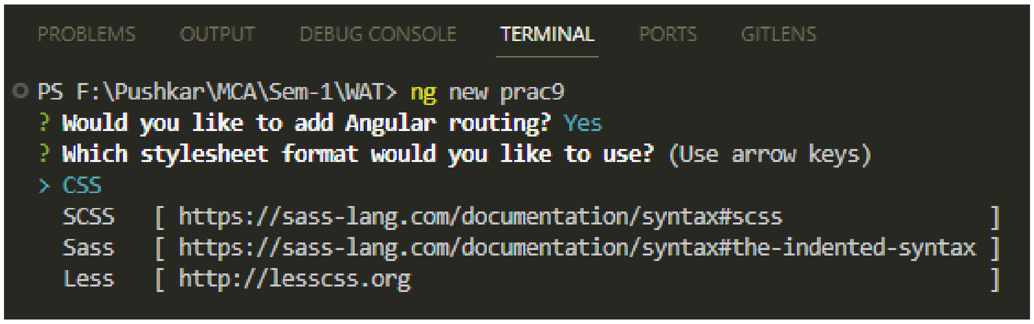
```
PS F:\Pushkar\MCA\Sem-1\WAT> ng new prac9
```

5. When prompted for Angular Routing, Enter "y"



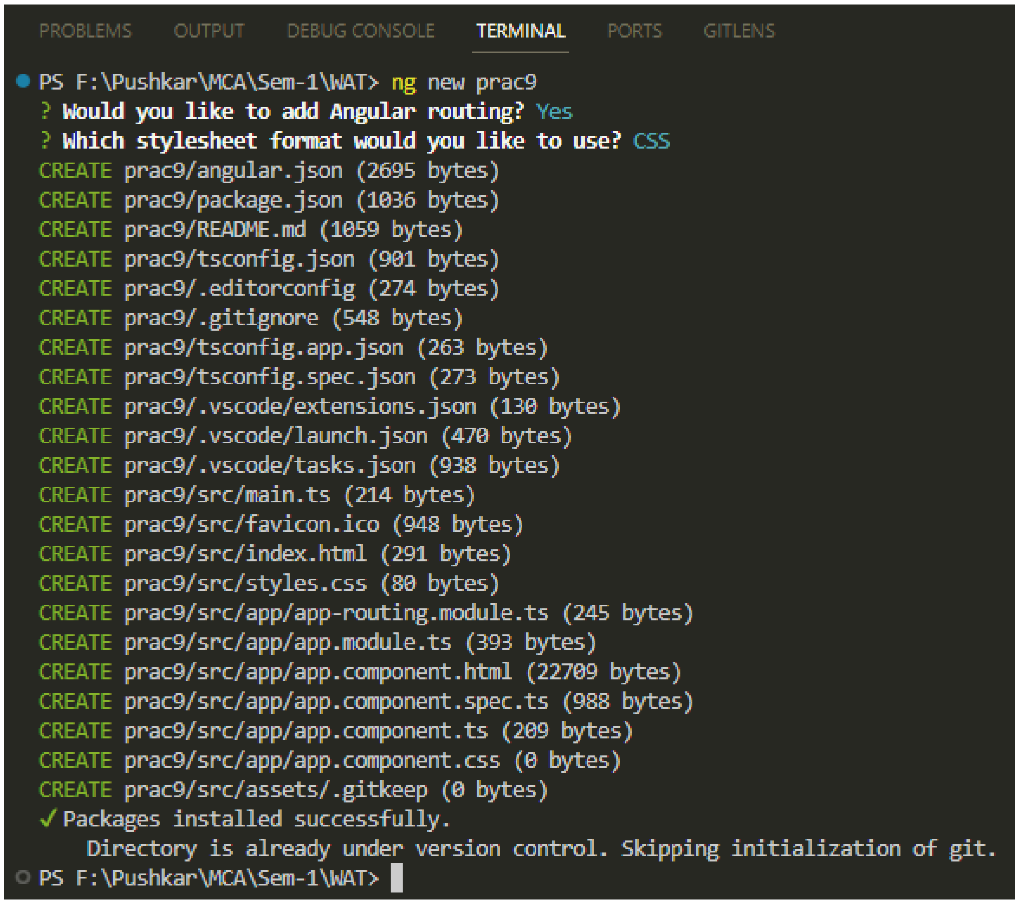
```
PS F:\Pushkar\MCA\Sem-1\WAT> ng new prac9
? Would you like to add Angular routing? (y/N) Y
```

6. When prompted for CSS, use arrow keys to select "CSS" (it is the first option and is selected by default) and then press Enter



```
PS F:\Pushkar\MCA\Sem-1\WAT> ng new prac9
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
```

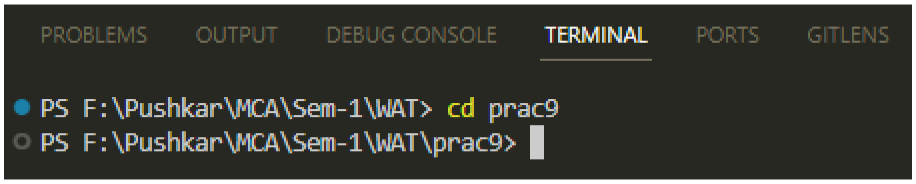
7. The new Angular project is created



```
PS F:\Pushkar\MCA\Sem-1\WAT> ng new prac9
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE prac9/angular.json (2695 bytes)
CREATE prac9/package.json (1036 bytes)
CREATE prac9/README.md (1059 bytes)
CREATE prac9/tsconfig.json (901 bytes)
CREATE prac9/.editorconfig (274 bytes)
CREATE prac9/.gitignore (548 bytes)
CREATE prac9/tsconfig.app.json (263 bytes)
CREATE prac9/tsconfig.spec.json (273 bytes)
CREATE prac9/.vscode/extensions.json (130 bytes)
CREATE prac9/.vscode/launch.json (470 bytes)
CREATE prac9/.vscode/tasks.json (938 bytes)
CREATE prac9/src/main.ts (214 bytes)
CREATE prac9/src/favicon.ico (948 bytes)
CREATE prac9/src/index.html (291 bytes)
CREATE prac9/src/styles.css (80 bytes)
CREATE prac9/src/app/app-routing.module.ts (245 bytes)
CREATE prac9/src/app/app.module.ts (393 bytes)
CREATE prac9/src/app/app.component.html (22709 bytes)
CREATE prac9/src/app/app.component.spec.ts (988 bytes)
CREATE prac9/src/app/app.component.ts (209 bytes)
CREATE prac9/src/app/app.component.css (0 bytes)
CREATE prac9/src/assets/.gitkeep (0 bytes)
✓ Packages installed successfully.
    Directory is already under version control. Skipping initialization of git.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

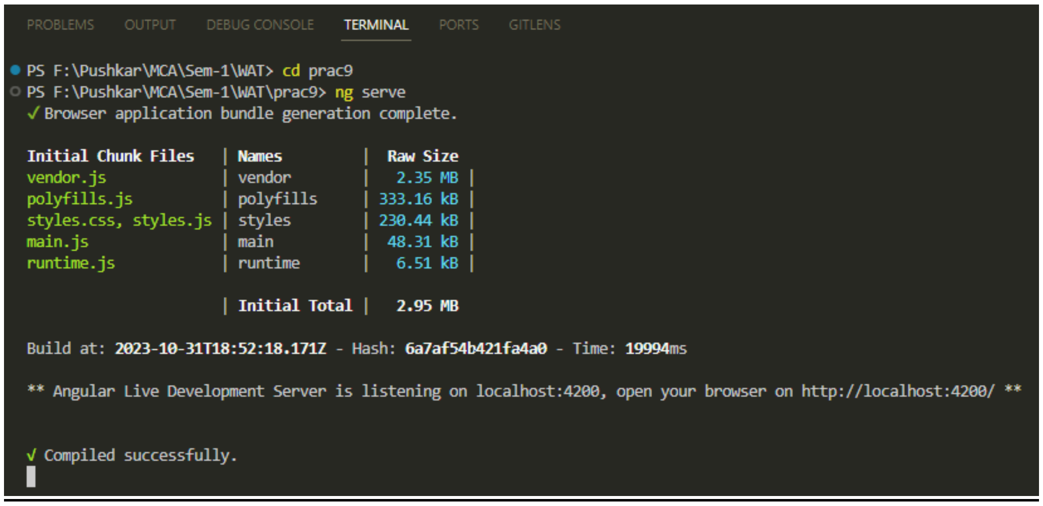
8. Navigate to the prac9 folder by using the cd command

cd prac9



```
PS F:\Pushkar\MCA\Sem-1\WAT> cd prac9
PS F:\Pushkar\MCA\Sem-1\WAT\prac9>
```

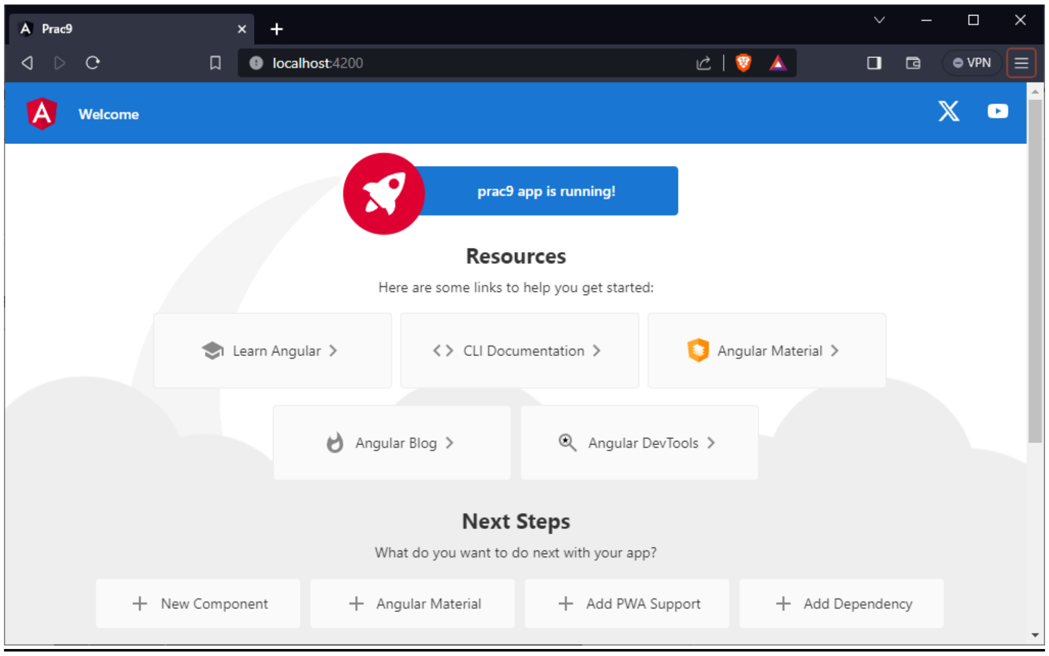
9. Run the following command to serve the angular project on a server
   ng serve



```
PS F:\Pushkar\MCA\Sem-1\WAT> cd prac9
PS F:\Pushkar\MCA\Sem-1\WAT\prac9> ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   2.35 MB |
polyfills.js          | polyfills     | 333.16 kB |
styles.css, styles.js | styles        | 230.44 kB |
main.js               | main          |  48.31 kB |
runtime.js            | runtime       |   6.51 kB |

                      | Initial Total |   2.95 MB

Build at: 2023-10-31T18:52:18.171Z - Hash: 6a7af54b421fa4a0 - Time: 19994ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
```

10. The server is running and will automatically re-compile the project when we make any changes.

11. Open any browser and go to the following link to see the output (this is the default output whenever any new Angular project is made)
    http://localhost:4200/

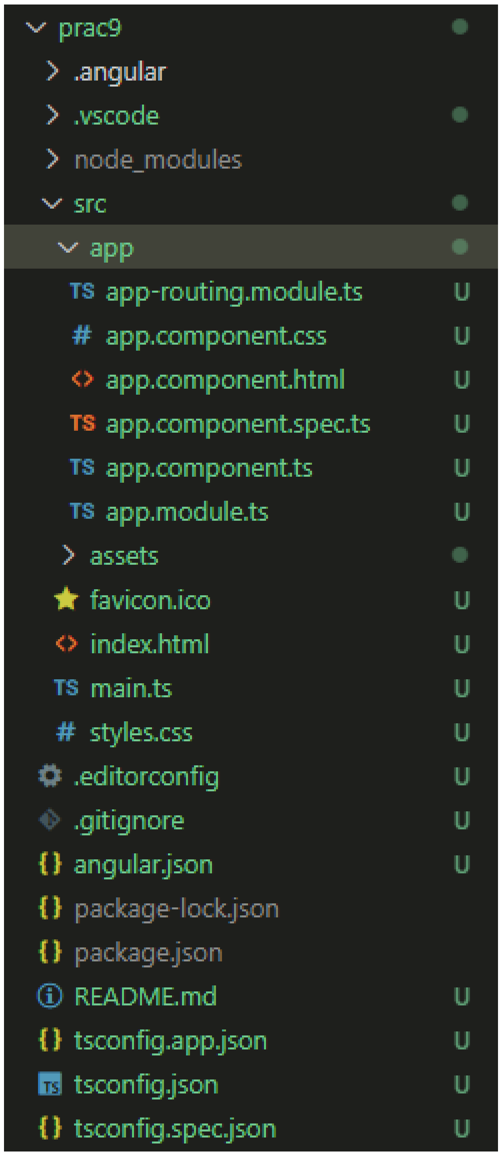


12. We will remove the default code that's generated and add our practical code.

    We will make changes to the following 2 files in the "app" folder (prac9/scr/app/)

    app.component.html

    app.component.ts



```
prac9
  .angular
  .vscode
  node_modules
  src
    app
      app-routing.module.ts      U
      app.component.css          U
      app.component.html         U
      app.component.spec.ts      U
      app.component.ts           U
      app.module.ts              U
    assets
    favicon.ico                  U
    index.html                   U
    main.ts                      U
    styles.css                   U
  .editorconfig                  U
  .gitignore                     U
  angular.json                   U
  package-lock.json
  package.json
  README.md                      U
  tsconfig.app.json              U
  tsconfig.json                  U
  tsconfig.spec.json             U
```

13. Open the **app.component.html** file and delete all the code
14. Write the following code in **app.component.html:**

    <!-- directive -->

    <!-- ngIf directive -->

    <!-- reference:- https://www.javatpoint.com/angular-8-ngif-directive -->

```
<h1> Directives </h1>
<h2> ngIf </h2>
n is = {{n}}
<div *ngIf="n > 2; else elseBlock">
  n is greater than 2
</div>

<ng-template #elseBlock>
  n is not greater than 2
</ng-template>

<!-- ngFor directive -->
<!-- reference:- https://www.javatpoint.com/angular-8-ngfor-directive -->
<h2> ngFor </h2>

<li *ngFor="let item of items">
  {{item}}
</li>

<!-- piping -->
<!-- reference:- https://www.tutorialspoint.com/angular8/angular8_pipes.htm -->
<h1> Piping </h1>
<h2> Adding parameter </h2>

<!-- adding parameter  -->
<div>
  Today's date :- {{presentDate}}
</div>

<div>
  Date with uppercase :- {{presentDate | date:'fullDate' | uppercase}}
<br />
  Date with lowercase :- {{presentDate | date:'medium' | lowercase}}
```

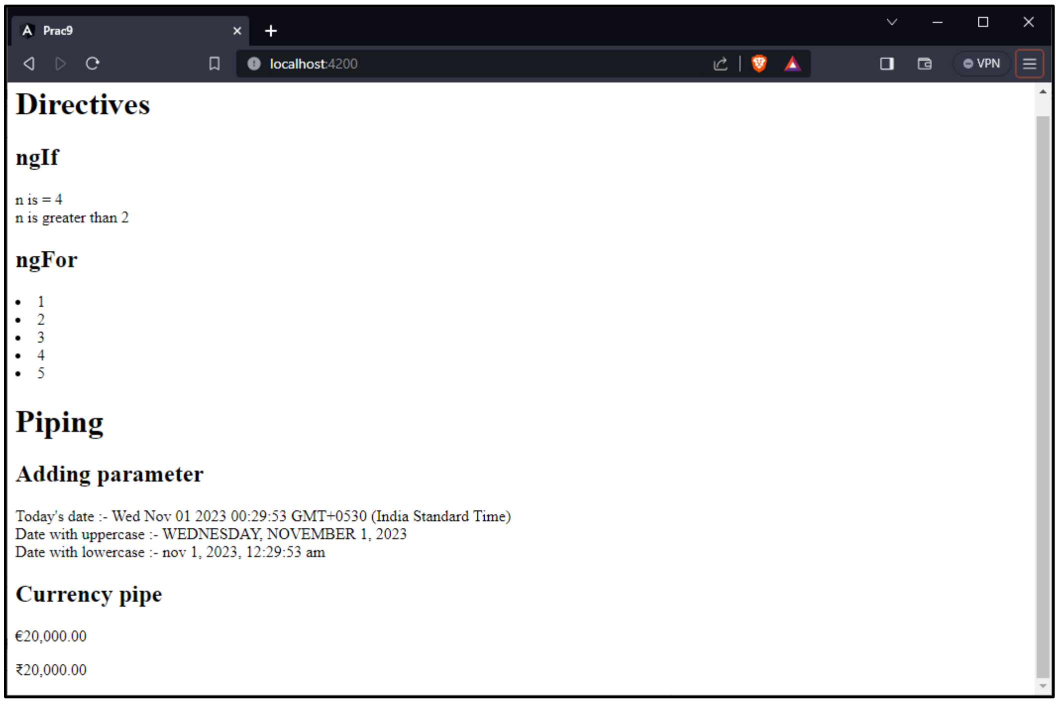
```
<br />
</div>

<!-- currency pipe -->
<h2> Currency pipe </h2>

<div>
  <p>{{ amount | currency:'EUR'}}</p>
  <p>{{ amount | currency:'INR' }}</p>
</div>
<router-outlet></router-outlet>
```

15. Save the **app.component.html** file.
16. Open the **app.component.ts** and make the following changes in it. The app.component.ts code should look like this:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'prac9';
  // added variables
  n: number = 4;
  presentDate: Date = new Date();
  amount: number = 20000;
  items: number[] = [1, 2, 3, 4, 5];
}
```

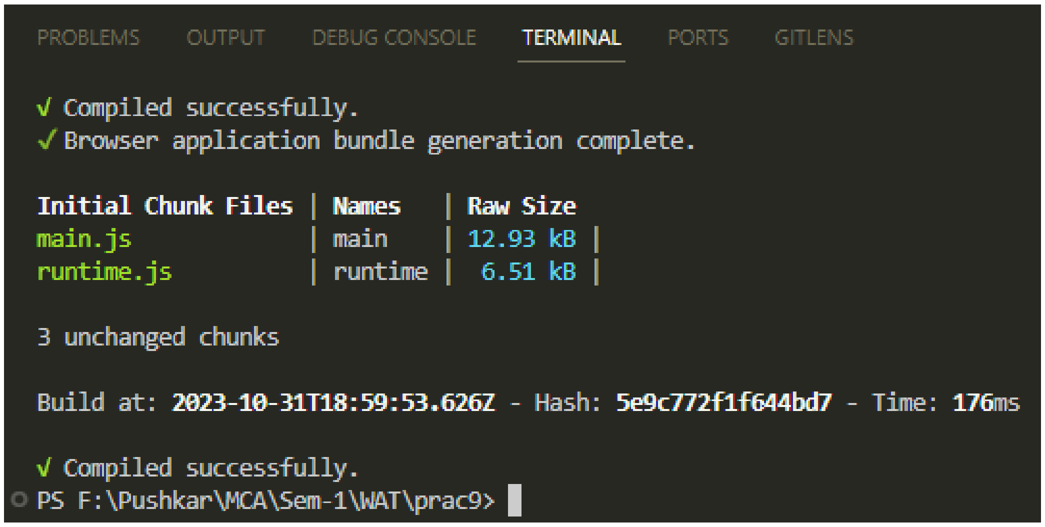
17. Save the **app.component.ts** file

18. Go to the browser to see the output at **http://localhost:4200/**



```
Directives

ngIf

n is = 4
n is greater than 2

ngFor

• 1
• 2
• 3
• 4
• 5

Piping

Adding parameter

Today's date :- Wed Nov 01 2023 00:29:53 GMT+0530 (India Standard Time)
Date with uppercase :- WEDNESDAY, NOVEMBER 1, 2023
Date with lowercase :- nov 1, 2023, 12:29:53 am

Currency pipe

€20,000.00

₹20,000.00
```

19. Go to the terminal and stop the server by pressing Ctrl + C
When we see the prompt as "....\prac9>" then it means that the server has stopped.



```
√ Compiled successfully.
√ Browser application bundle generation complete.

Initial Chunk Files | Names   | Raw Size
main.js             | main    | 12.93 kB |
runtime.js          | runtime |  6.51 kB |

3 unchanged chunks

Build at: 2023-10-31T18:59:53.626Z - Hash: 5e9c772f1f644bd7 - Time: 176ms

√ Compiled successfully.
PS F:\Pushkar\MCA\Sem-1\WAT\prac9>
```

Code:

app.component.html

```
<!-- directive -->
<!-- ngIf directive -->
<!-- reference:- https://www.javatpoint.com/angular-8-ngif-directive -->

<h1> Directives </h1>

<h2> ngIf </h2>

n is = {{n}}

<div *ngIf="n > 2; else elseBlock">
  n is greater than 2
</div>

<ng-template #elseBlock>
  n is not greater than 2
</ng-template>

<!-- ngFor directive -->
<!-- reference:- https://www.javatpoint.com/angular-8-ngfor-directive -->

<h2> ngFor </h2>
<li *ngFor="let item of items">
  {{item}}
</li>

<!-- piping -->
<!-- reference:- https://www.tutorialspoint.com/angular8/angular8_pipes.htm -->

<h1> Piping </h1>
<h2> Adding parameter </h2>
```

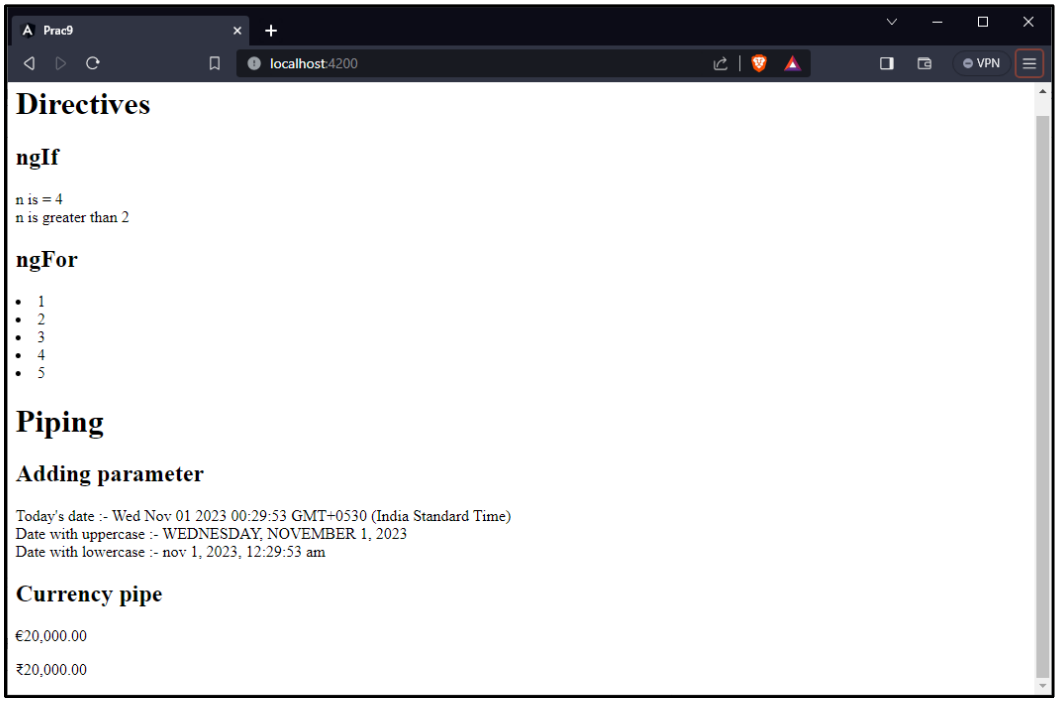
```
<!-- adding parameter  -->
<div>
  Today's date :- {{presentDate}}
</div>

<div>
  Date with uppercase :- {{presentDate | date:'fullDate' | uppercase}}
<br />
  Date with lowercase :- {{presentDate | date:'medium' | lowercase}}
<br />
</div>

<!-- currency pipe -->
<h2> Currency pipe </h2>

<div>
  <p>{{ amount | currency:'EUR'}}</p>
  <p>{{ amount | currency:'INR' }}</p>
</div>
<router-outlet></router-outlet>
```

app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'prac9';
  // added variables
  n: number = 4;
```

```
    presentDate: Date = new Date();
    amount: number = 20000;
    items: number[] = [1, 2, 3, 4, 5];
  }
```

Output:



```
Directives

ngIf

n is = 4
n is greater than 2

ngFor

•  1
•  2
•  3
•  4
•  5

Piping

Adding parameter

Today's date :- Wed Nov 01 2023 00:29:53 GMT+0530 (India Standard Time)
Date with uppercase :- WEDNESDAY, NOVEMBER 1, 2023
Date with lowercase :- nov 1, 2023, 12:29:53 am

Currency pipe

€20,000.00

₹20,000.00
```

Conclusion: We've explored an introduction to Node.JS, Advantages and Disadvantages, Node.js Process Model, Traditional Web Server Model, Installation and some programs with node.js.

Name of Student: Pushkar Sane			
Roll Number: 45		**Practical Number: 10**	
Title of Lab Assignment: Demonstrate features of Angular forms with a Program.			
DOP: 23-09-2023		**DOS: 24-09-2023**	
CO Mapped: **CO5**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No. 10

Aim: Demonstrate features of Angular forms with a Program.

Description:
For reference: YouTube playlists for explanation of the code

Hindi: https://www.youtube.com/playlist?list=PL8p2I9GklV45--5t7_N4lveUI6Y31vQ6C
(Watch videos #35 to #37)

OR

English: https://www.youtube.com/playlist?list=PL8p2I9GklV47eNpoo4Fr6fkags72a8F0v
(Watch videos #34 to #37)

Angular is a popular web application framework that provides a variety of features for building dynamic and interactive web applications. Angular forms are a critical part of this framework, allowing developers to capture and handle user input. Here are some of the key features of Angular forms:

1. **Template-driven forms:**
   Angular offers a template-driven approach for creating forms, which is suitable for simpler forms with less complex logic.
   Forms are defined within the template HTML and are bound to the component using ngModel directives.

2. **Reactive forms (Model-driven forms):**
   Reactive forms are more flexible and suitable for complex forms with extensive validation and custom logic.
   They are defined programmatically in the component class, offering full control over form behavior and validation.

3. **Two-way data binding:**
   In template-driven forms, Angular provides two-way data binding using ngModel, allowing data to flow seamlessly between the template and component.

4. **FormControl and FormGroup:**
   Reactive forms use FormControl and FormGroup classes to represent form controls and groups of controls.
   FormControl represents an individual input field, while FormGroup is used to group related form controls together.

5. **Validation:**
   Both template-driven and reactive forms support client-side validation with built-in validators like required, minLength, maxLength, pattern, and custom validators.
   Validation messages can be displayed based on the form's state and user interactions.

6. **Asynchronous validation:**
   Reactive forms allow you to perform asynchronous validation, such as checking data on a server or making HTTP requests before allowing form submission.

7. **Dynamic form controls:**
   You can dynamically add, remove, or modify form controls at runtime based on user interactions or other criteria.

8. **FormArray:**
   Reactive forms include FormArray, which allows you to work with dynamic lists of form controls, like repeating form fields or dynamic form sections.

9. **Error handling:**
   Angular provides mechanisms for handling and displaying form errors and validation messages, making it easier for users to understand and correct input issues.

10. **Form submission:**
    Angular forms provide mechanisms for handling form submission, whether it's sending data to a server via HTTP requests or performing actions on the client side.

11. ngSubmit:

Both template-driven and reactive forms support the ngSubmit directive, which allows you to define a function to be executed when the form is submitted.

12. Template-driven form features:

Features specific to template-driven forms include ngModel, ngModelGroup, ngForm, and ngModelChange for custom event handling.

13. Reactive form features:

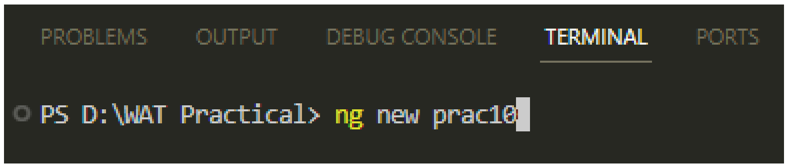
Features specific to reactive forms include FormBuilder for simplifying form control creation, custom validators, value changes observables, and more control over the form's behavior.

1. **Demonstrate features of Angular forms with a program**

   Steps to execute the practical:

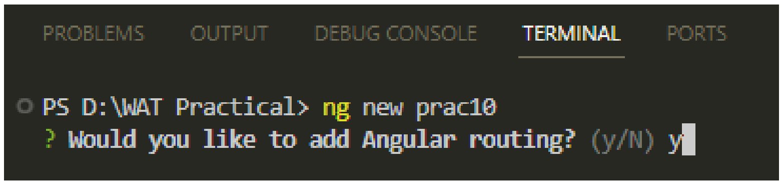
   1. Create a folder “practical 10”
   2. Open the terminal (please use cmd, avoid using powershell) and navigate the “practical 10” folder by using the cd command
   3. Create a new angular project by running the following command

      ng new prac10



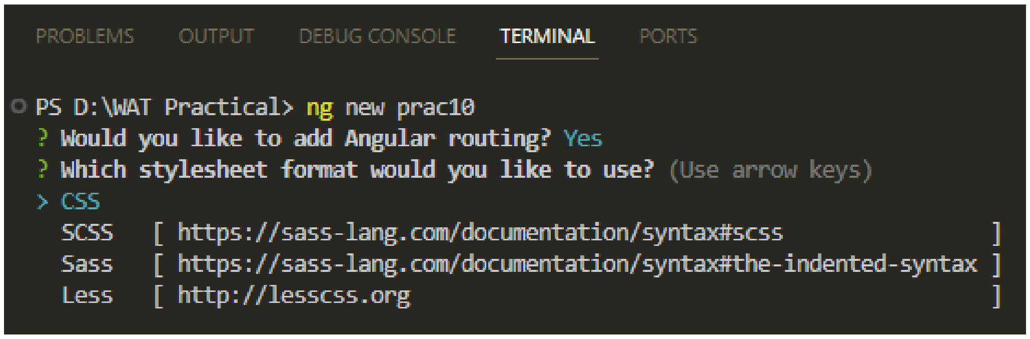
```
PS D:\WAT Practical> ng new prac10
```

   4. When prompted for Angular Routing, Enter “y”



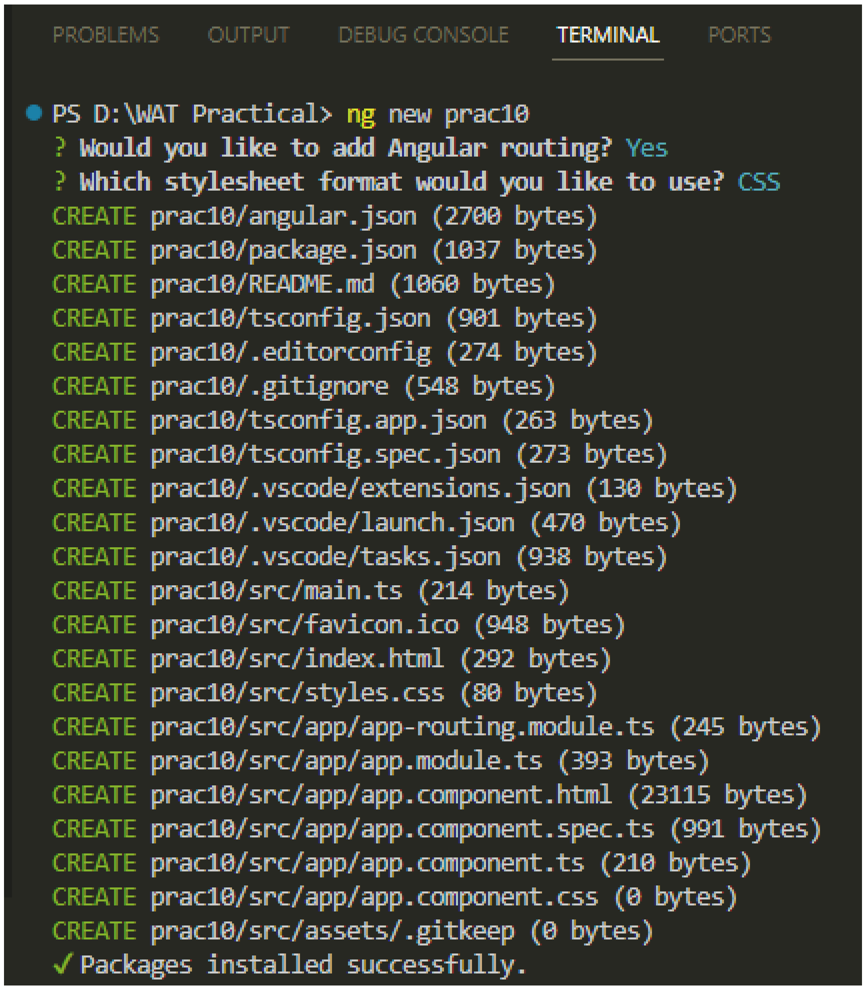
```
PS D:\WAT Practical> ng new prac10
? Would you like to add Angular routing? (y/N) y
```

5. When prompted for CSS, use arrow keys to select “CSS” (it is the first option and is selected by default) and then press Enter



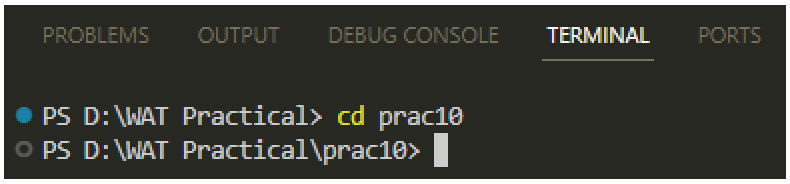
```
PS D:\WAT Practical> ng new prac10
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
```

6. The new Angular project is created



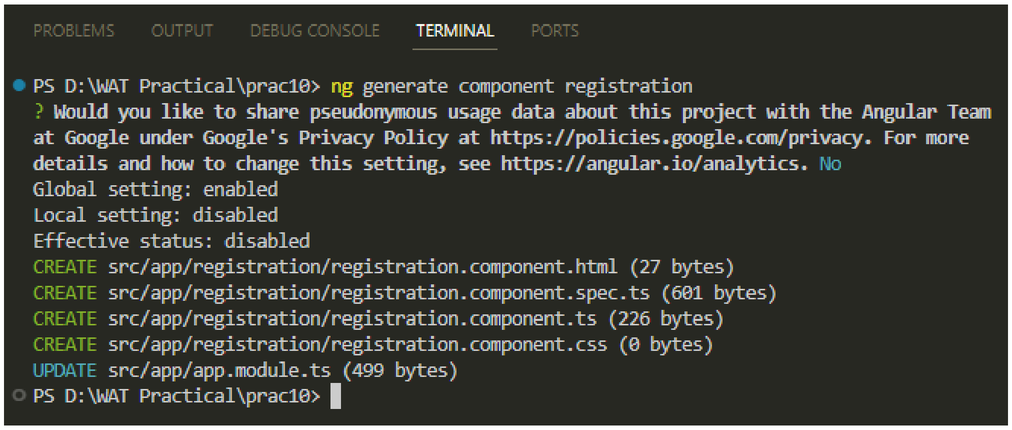
```
PS D:\WAT Practical> ng new prac10
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE prac10/angular.json (2700 bytes)
CREATE prac10/package.json (1037 bytes)
CREATE prac10/README.md (1060 bytes)
CREATE prac10/tsconfig.json (901 bytes)
CREATE prac10/.editorconfig (274 bytes)
CREATE prac10/.gitignore (548 bytes)
CREATE prac10/tsconfig.app.json (263 bytes)
CREATE prac10/tsconfig.spec.json (273 bytes)
CREATE prac10/.vscode/extensions.json (130 bytes)
CREATE prac10/.vscode/launch.json (470 bytes)
CREATE prac10/.vscode/tasks.json (938 bytes)
CREATE prac10/src/main.ts (214 bytes)
CREATE prac10/src/favicon.ico (948 bytes)
CREATE prac10/src/index.html (292 bytes)
CREATE prac10/src/styles.css (80 bytes)
CREATE prac10/src/app/app-routing.module.ts (245 bytes)
CREATE prac10/src/app/app.module.ts (393 bytes)
CREATE prac10/src/app/app.component.html (23115 bytes)
CREATE prac10/src/app/app.component.spec.ts (991 bytes)
CREATE prac10/src/app/app.component.ts (210 bytes)
CREATE prac10/src/app/app.component.css (0 bytes)
CREATE prac10/src/assets/.gitkeep (0 bytes)
✓ Packages installed successfully.
```

7. Navigate to the prac10 folder by using the cd command
   cd prac10



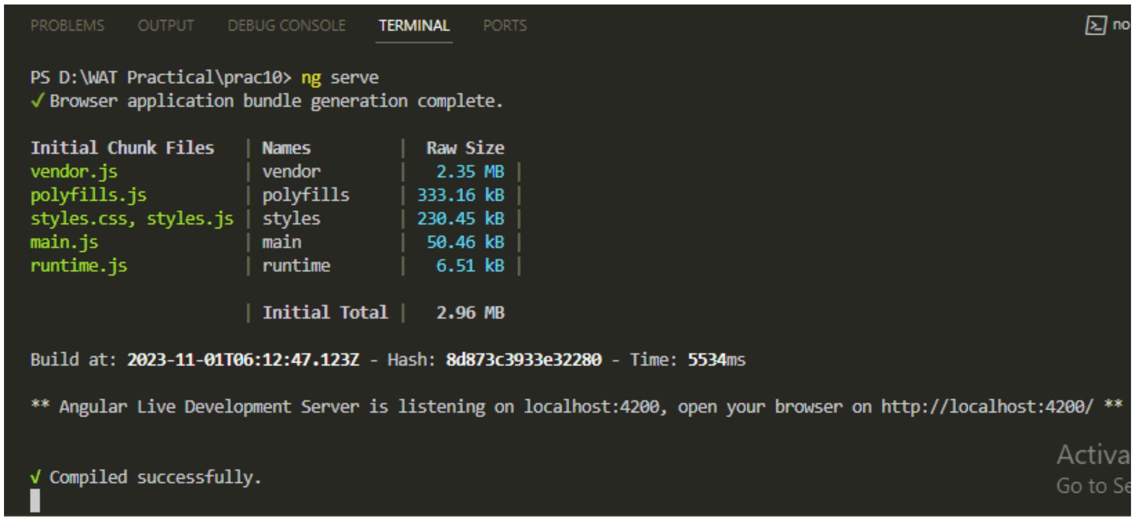
```
PS D:\WAT Practical> cd prac10
PS D:\WAT Practical\prac10>
```

8. Create a new component “registration” by running the following command. This registration component will be our registration form
   ng generate component registration



```
PS D:\WAT Practical\prac10> ng generate component registration
? 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.io/analytics. No
Global setting: enabled
Local setting: disabled
Effective status: disabled
CREATE src/app/registration/registration.component.html (27 bytes)
CREATE src/app/registration/registration.component.spec.ts (601 bytes)
CREATE src/app/registration/registration.component.ts (226 bytes)
CREATE src/app/registration/registration.component.css (0 bytes)
UPDATE src/app/app.module.ts (499 bytes)
PS D:\WAT Practical\prac10>
```

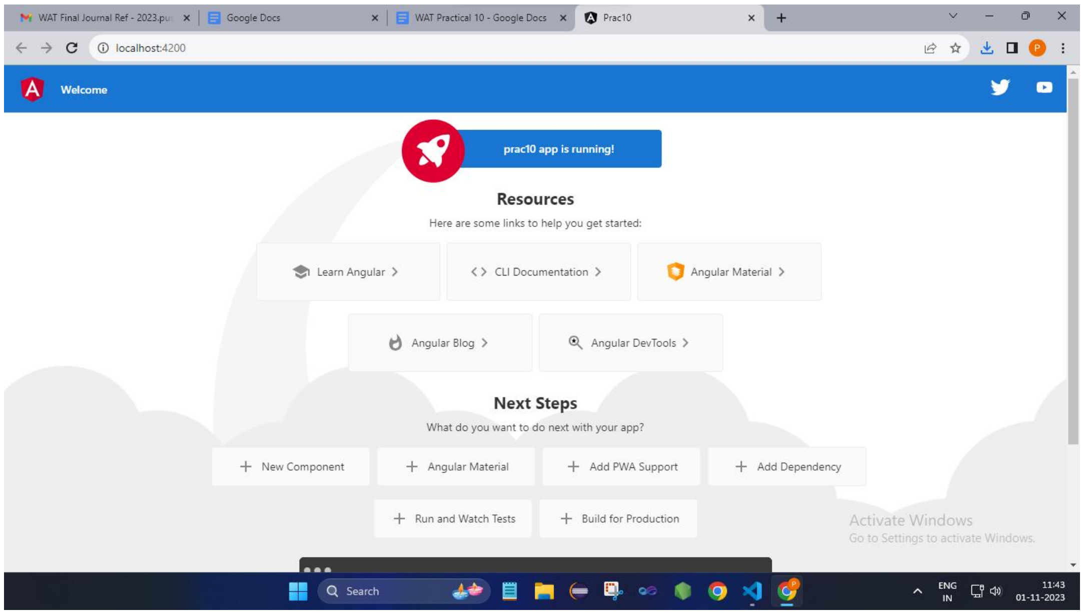
9. Run the following command to serve the angular project on a server
   ng serve



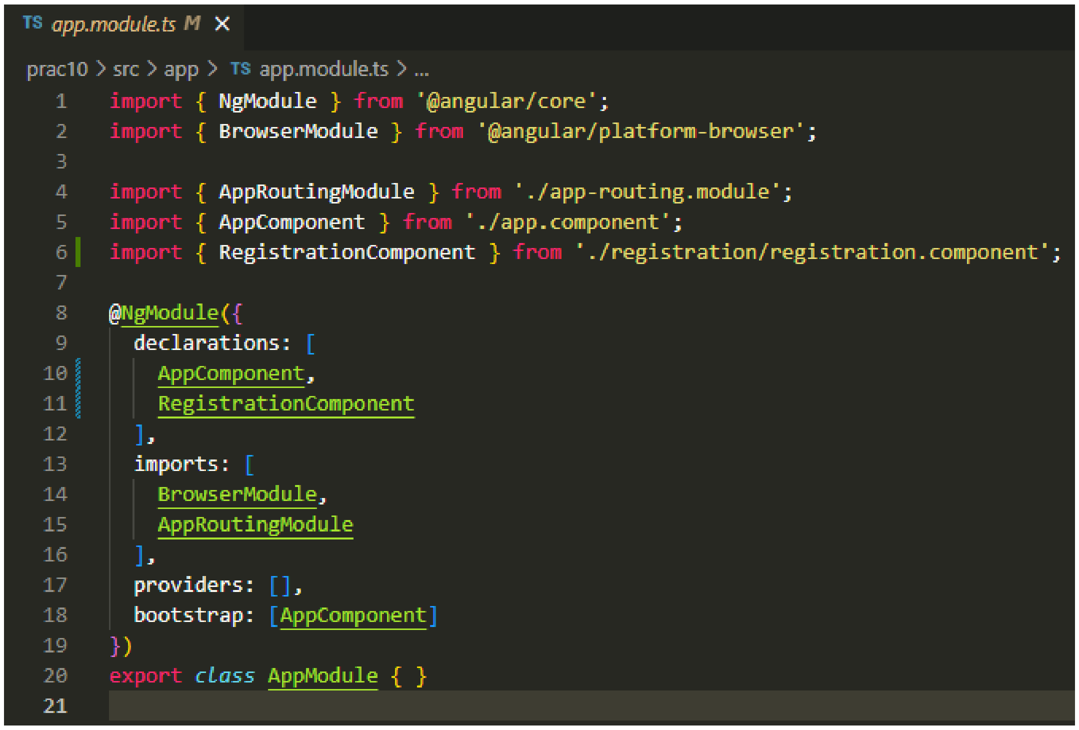
```
PS D:\WAT Practical\prac10> ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   2.35 MB |
polyfills.js          | polyfills     | 333.16 kB |
styles.css, styles.js | styles        | 230.45 kB |
main.js               | main          |  50.46 kB |
runtime.js            | runtime       |   6.51 kB |

                      | Initial Total |   2.96 MB

Build at: 2023-11-01T06:12:47.123Z - Hash: 8d873c3933e32280 - Time: 5534ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
```

10. The server is running and will automatically re-compile the project when we make any changes

11. Open any browser and go to the following link to see the output (this is the default output whenever any new Angular project is made)
    http://localhost:4200/



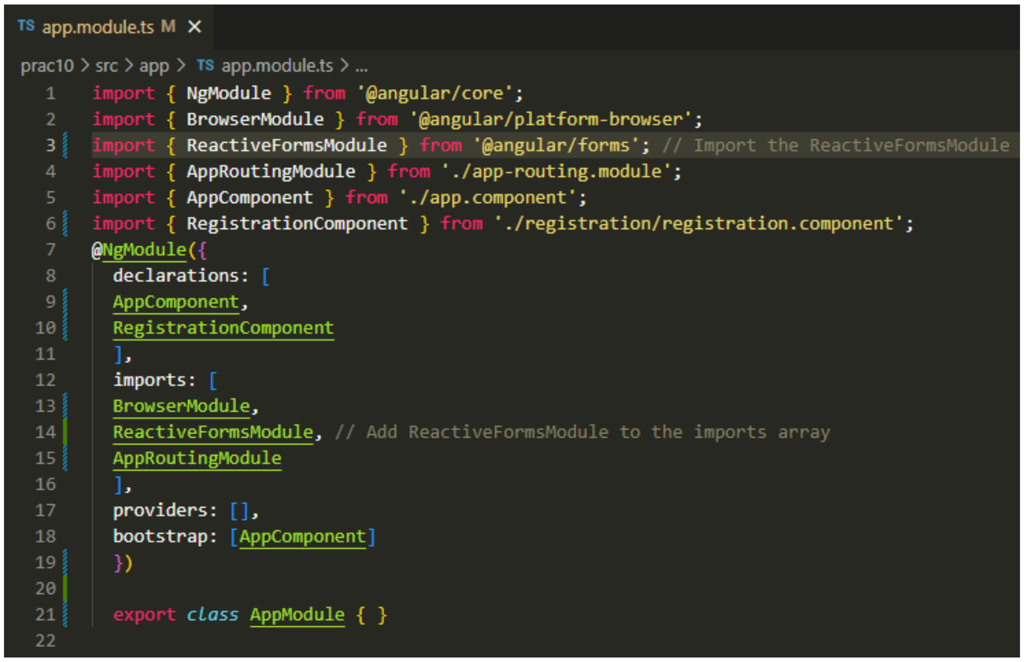
12. We will add the ReactiveFormsModule to the **app.module.t**s file in the prac10/src/app/ folder



app.module.ts M ×

prac10 > src > app > app.module.ts > ...

```typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { RegistrationComponent } from './registration/registration.component';

@NgModule({
  declarations: [
    AppComponent,
    RegistrationComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



app.module.ts M ×

prac10 > src > app > app.module.ts > ...

```typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ReactiveFormsModule } from '@angular/forms'; // Import the ReactiveFormsModule
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { RegistrationComponent } from './registration/registration.component';
@NgModule({
  declarations: [
  AppComponent,
  RegistrationComponent
  ],
  imports: [
  BrowserModule,
  ReactiveFormsModule, // Add ReactiveFormsModule to the imports array
  AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
  })

  export class AppModule { }
```

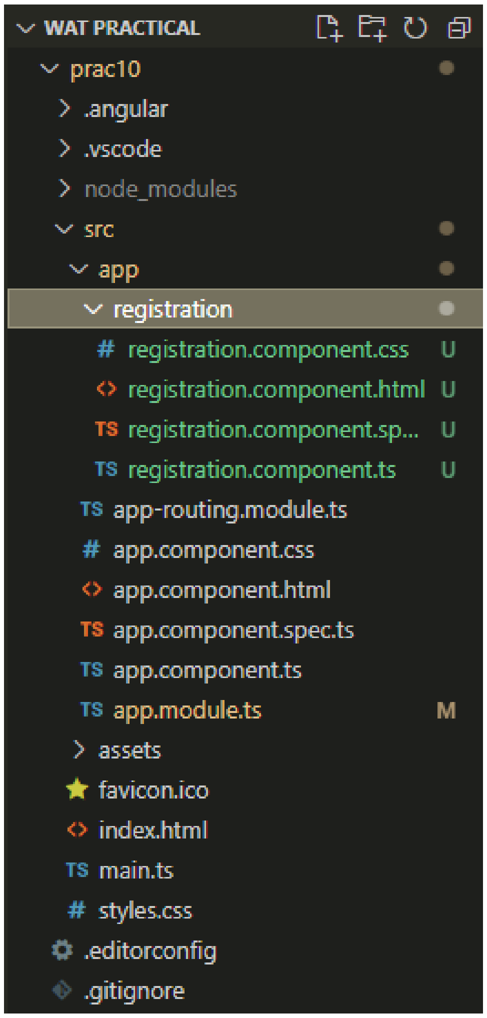
After making the changes, the **app.module.ts** file should look like this:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ReactiveFormsModule } from '@angular/forms'; // Import the
ReactiveFormsModule
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { RegistrationComponent } from './registration/registration.component';
@NgModule({
  declarations: [
  AppComponent,
  RegistrationComponent
  ],
  imports: [
  BrowserModule,
  ReactiveFormsModule, // Add ReactiveFormsModule to the imports array
  AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
  })
  export class AppModule { }
```

13. Add the “registration” component to the app.component.html file
    Delete all the contents of **app.component.html** file (located in the prac11/src/app/ folder)
    Add the following code to the file and save it
    <app-registration></app-registration>

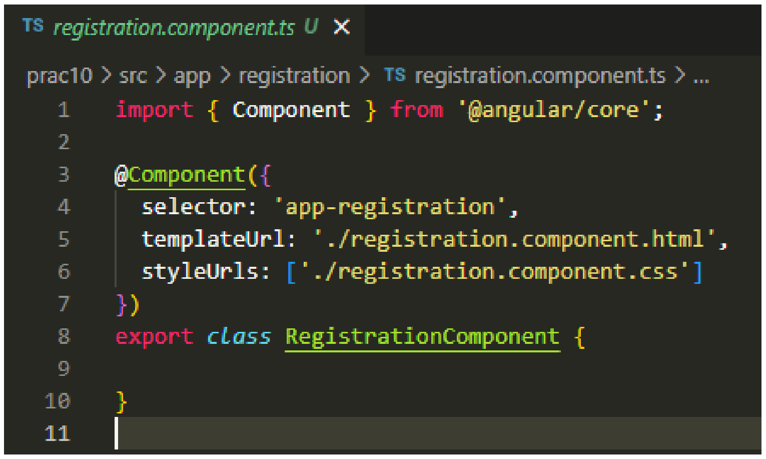
14. Now we’ll make the registration form.
    We will make changes to the following 3 files in the “registration” folder (prac10/src/app/registration/)
    registration.component.ts
    registration.component.html

registration.component.css



```
WAT PRACTICAL
  prac10
    > .angular
    > .vscode
    > node_modules
    src
      app
        registration
          # registration.component.css      U
          <> registration.component.html    U
          TS registration.component.sp...   U
          TS registration.component.ts      U
        TS app-routing.module.ts
        # app.component.css
        <> app.component.html
        TS app.component.spec.ts
        TS app.component.ts
        TS app.module.ts                     M
      > assets
      favicon.ico
      <> index.html
      TS main.ts
      # styles.css
    .editorconfig
    .gitignore
```

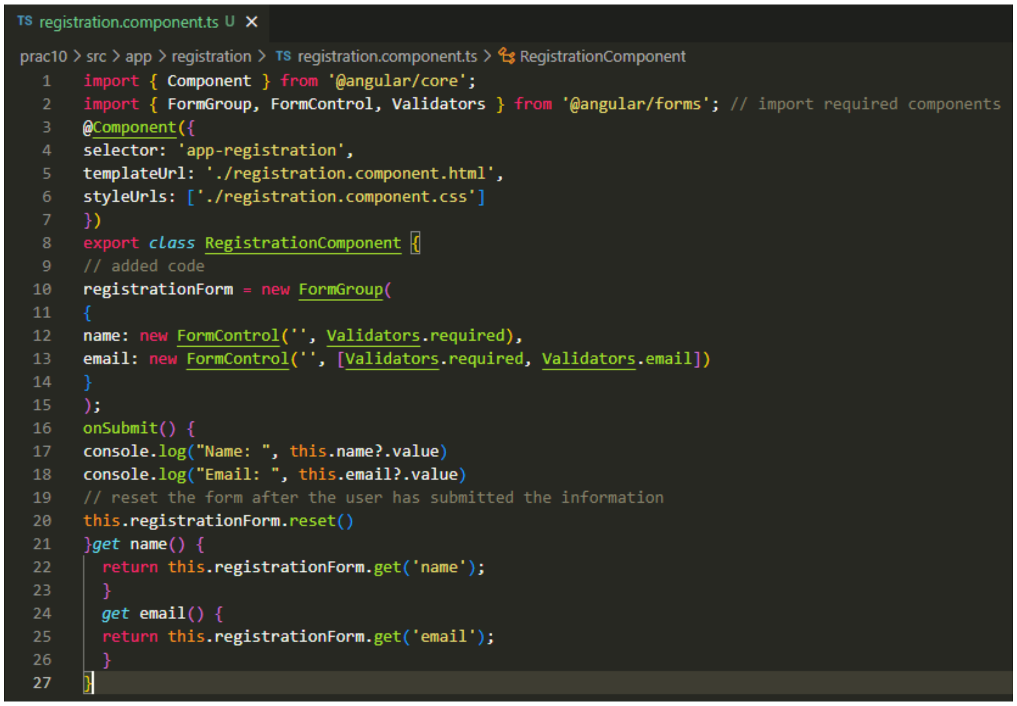
15. We'll modify the registration.component.ts file in the prac10/src/app/registration folder



registration.component.ts U

prac10 > src > app > registration > registration.component.ts > ...

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

@Component({
  selector: 'app-registration',
  templateUrl: './registration.component.html',
  styleUrls: ['./registration.component.css']
})
export class RegistrationComponent {

}
```



```
registration.component.ts U ×
prac10 > src > app > registration > registration.component.ts > RegistrationComponent
import { Component } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms'; // import required components
@Component({
selector: 'app-registration',
templateUrl: './registration.component.html',
styleUrls: ['./registration.component.css']
})
export class RegistrationComponent {
// added code
registrationForm = new FormGroup(
{
name: new FormControl('', Validators.required),
email: new FormControl('', [Validators.required, Validators.email])
}
);
onSubmit() {
console.log("Name: ", this.name?.value)
console.log("Email: ", this.email?.value)
// reset the form after the user has submitted the information
this.registrationForm.reset()
}get name() {
  return this.registrationForm.get('name');
  }
  get email() {
  return this.registrationForm.get('email');
  }
}
```

After making the changes, the **registration.component.ts** file should look like this:

import { Component } from '@angular/core';

import { FormGroup, FormControl, Validators } from '@angular/forms'; // import required components

@Component({

selector: 'app-registration',

templateUrl: './registration.component.html',

styleUrls: ['./registration.component.css']

})

export class RegistrationComponent {

// added code

registrationForm = new FormGroup(

{

name: new FormControl('', Validators.required),

email: new FormControl('', [Validators.required, Validators.email])

}

);

onSubmit() {

```
console.log("Name: ", this.name?.value)
console.log("Email: ", this.email?.value)
// reset the form after the user has submitted the information
this.registrationForm.reset()
}get name() {
  return this.registrationForm.get('name');
  }
  get email() {
  return this.registrationForm.get('email');
  }
}
```

16. Delete all the default generated code in registration.component.html file in the prac10/src/app/registration folder.
    Write the following code in registration.component.html

```
<h1>Registration Form</h1>
<form [formGroup]="registrationForm" (ngSubmit)="onSubmit()">
    Name: <input type="text" formControlName="name">
    <p class="error-message" *ngIf="name?.touched && name?.invalid">Name is
invalid</p>
    <br>
    Email: <input type="email" formControlName="email">
    <p class="error-message" *ngIf="email?.touched && email?.invalid">Email
isinvalid</p>
    <br>
    <button [disabled]="registrationForm.invalid">Submit</button>
</form>
```

17. Save the registration.component.html file.
18. Open the registration.component.css and add the following code in it:

```
.error-message {
color: red;
}
```

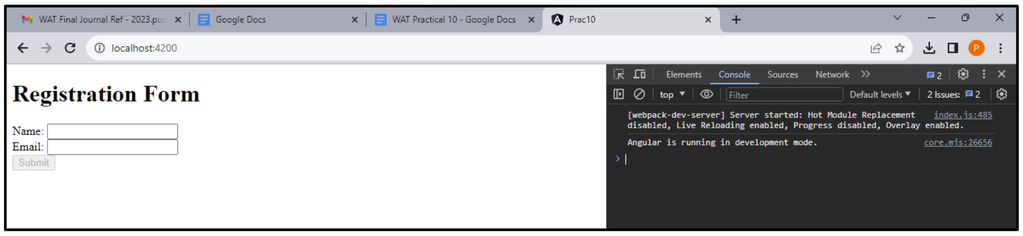
19. Save the registration.component.css file

20. Go to the browser to see the output at http://localhost:4200/

    Press the following keyboard shortcut to see the console in the browser

    Ctrl + Shift + J

    Registration Form Output

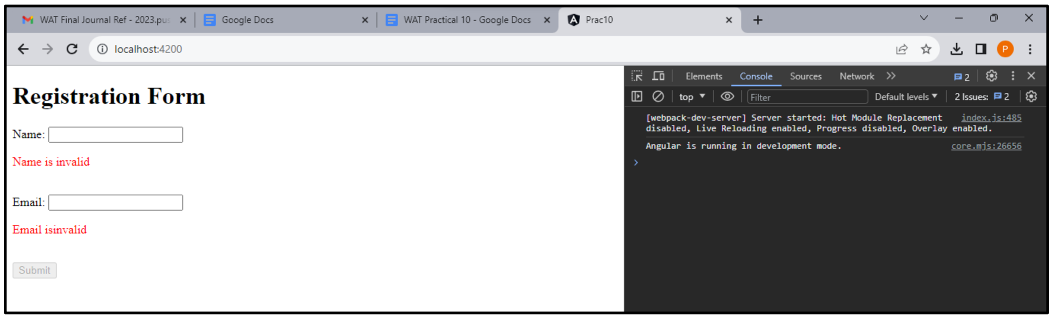


Registration Form

Name:
Email:
Submit

```
[webpack-dev-server] Server started: Hot Module Replacement     index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                         core.mjs:26656
```

Click on the input boxes and then click outside the boxes to see the form validation



Registration Form

Name:

Name is invalid

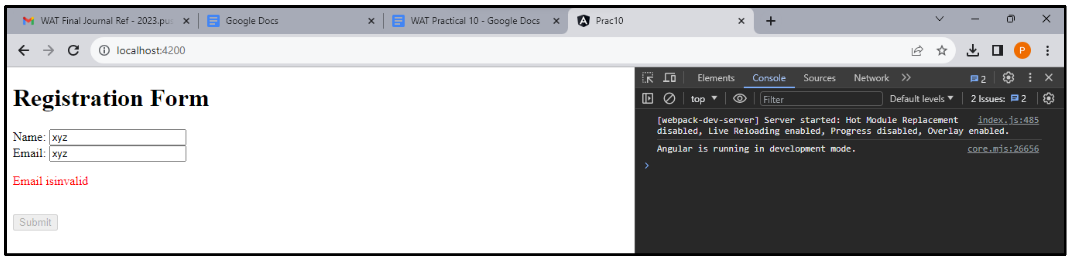
Email:

Email isinvalid

Submit

```
[webpack-dev-server] Server started: Hot Module Replacement     index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                         core.mjs:26656
```

Type all valid input values to enable the Submit Button



Registration Form

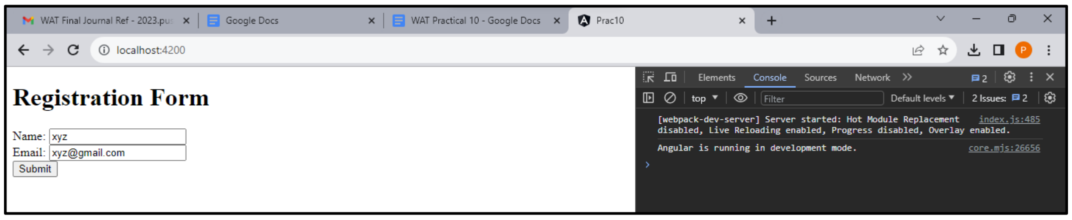
Name: xyz
Email: xyz

Email isinvalid

Submit

```
[webpack-dev-server] Server started: Hot Module Replacement     index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                         core.mjs:26656
```

Type all valid input values to enable the Submit Button

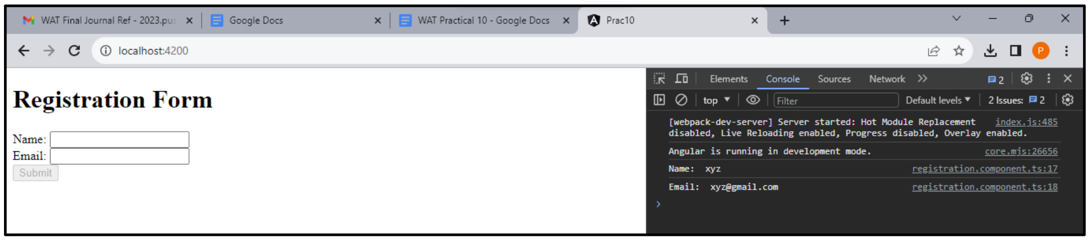


Registration Form

Name: xyz
Email: xyz@gmail.com
Submit

```
[webpack-dev-server] Server started: Hot Module Replacement     index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                         core.mjs:26656
```

Submit the form to see the values in the console



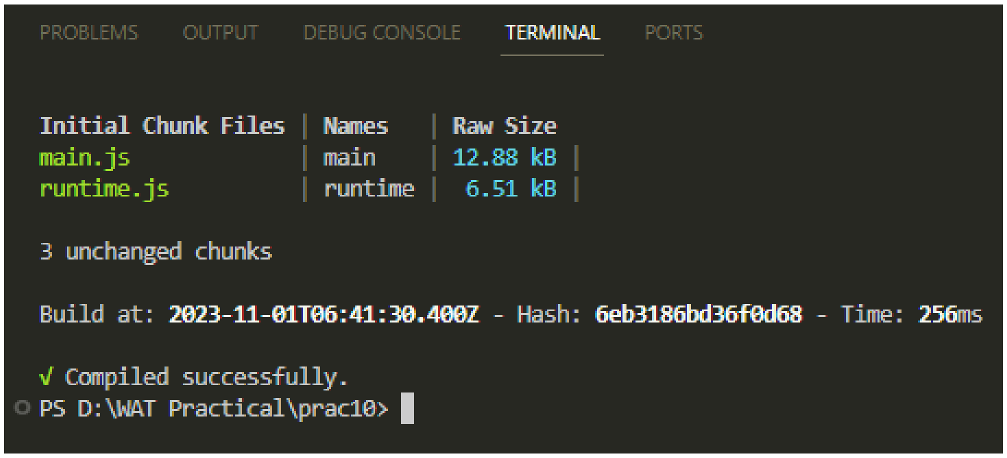
Registration Form

Name:
Email:
Submit

```
[webpack-dev-server] Server started: Hot Module Replacement    index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                        core.mjs:26656
Name:  xyz                                       registration.component.ts:17
Email:  xyz@gmail.com                            registration.component.ts:18
>
```

21. Go to the terminal and stop the server by pressing Ctrl + C

When we see the prompt as "....\prac10>" then it means that the server has stopped



```
Initial Chunk Files | Names    | Raw Size
main.js             | main     | 12.88 kB |
runtime.js          | runtime  |  6.51 kB |

3 unchanged chunks

Build at: 2023-11-01T06:41:30.400Z - Hash: 6eb3186bd36f0d68 - Time: 256ms

√ Compiled successfully.
PS D:\WAT Practical\prac10>
```

Code:

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ReactiveFormsModule } from '@angular/forms'; // Import the ReactiveFormsModule
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { RegistrationComponent } from './registration/registration.component';
@NgModule({
  declarations: [
  AppComponent,
  RegistrationComponent
  ],
  imports: [
  BrowserModule,
  ReactiveFormsModule, // Add ReactiveFormsModule to the imports array
  AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

```
<app-registration></app-registration>
<router-outlet></router-outlet>
```

registration.component.ts

```
import { Component } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms'; // import required
components
@Component({
selector: 'app-registration',
templateUrl: './registration.component.html',
styleUrls: ['./registration.component.css']
})
export class RegistrationComponent {
// added code
registrationForm = new FormGroup(
{
name: new FormControl('', Validators.required),
email: new FormControl('', [Validators.required, Validators.email])
}
);
onSubmit() {
console.log("Name: ", this.name?.value)
console.log("Email: ", this.email?.value)
// reset the form after the user has submitted the information
this.registrationForm.reset()
} get name() {
  return this.registrationForm.get('name');
  }
  get email() {
  return this.registrationForm.get('email');
  }
}
```

registration.component.html

```
<h1>Registration Form</h1>
<form [formGroup]="registrationForm" (ngSubmit)="onSubmit()">
    Name: <input type="text" formControlName="name">
    <p class="error-message" *ngIf="name?.touched && name?.invalid">Name is
invalid</p>
    <br>
    Email: <input type="email" formControlName="email">
    <p class="error-message" *ngIf="email?.touched && email?.invalid">Email
isinvalid</p>
    <br>
    <button [disabled]="registrationForm.invalid">Submit</button>
</form>
```

registration.component.css

```
.error-message {
    color: red;
}
```

Reference for explanation of the code

Hindi: https://www.youtube.com/playlist?list=PL8p2I9GklV45--5t7_N4lveUI6Y31vQ6C

(Watch videos #35 to #37)

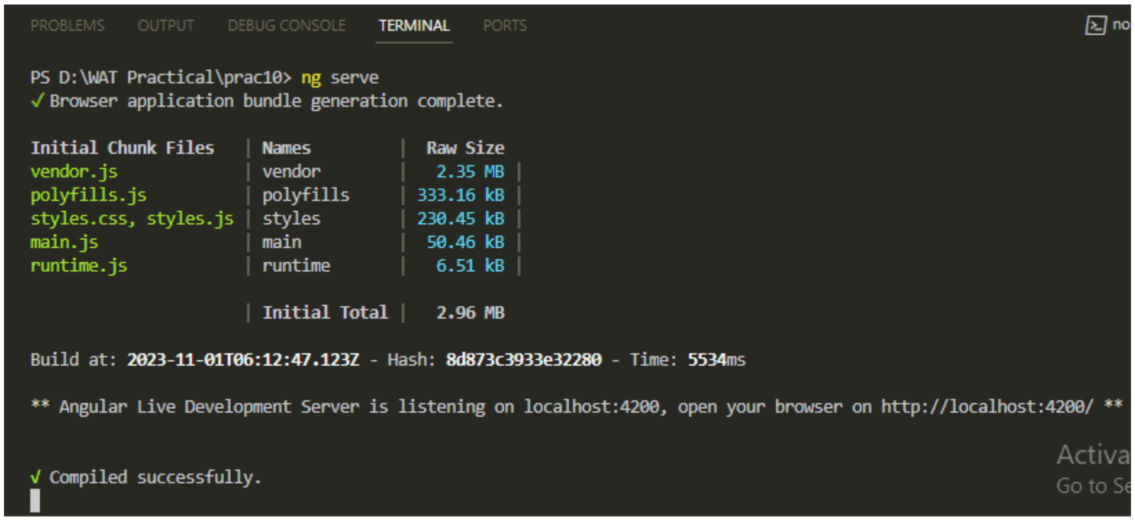
OR

English: https://www.youtube.com/playlist?list=PL8p2I9GklV47eNpoo4Fr6fkags72a8F0v

(Watch videos #34 to #37)

<u>Output:</u>

Console

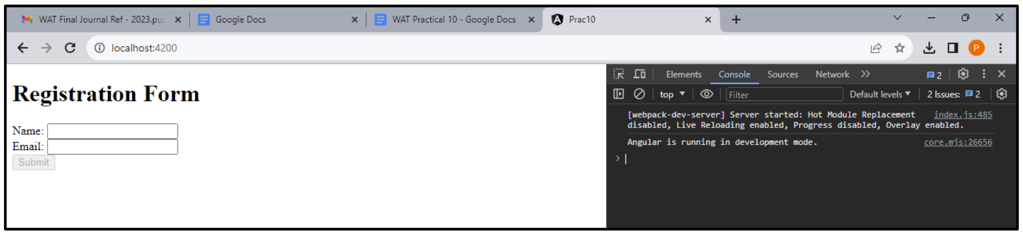


```
PS D:\WAT Practical\prac10> ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size |
vendor.js             | vendor        |   2.35 MB |
polyfills.js          | polyfills     | 333.16 kB |
styles.css, styles.js | styles        | 230.45 kB |
main.js               | main          |  50.46 kB |
runtime.js            | runtime       |   6.51 kB |

                      | Initial Total |   2.96 MB

Build at: 2023-11-01T06:12:47.123Z - Hash: 8d873c3933e32280 - Time: 5534ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
```

Browser:

Press Ctrl + Shift + J to see the console

Registration Form Output

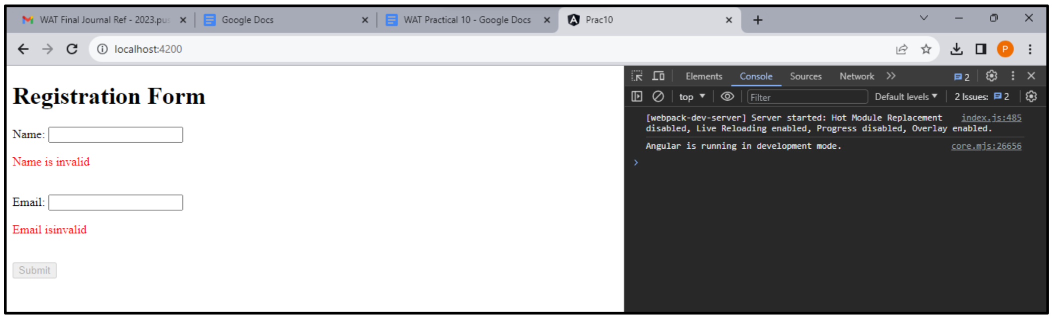


Registration Form

Name:
Email:

```
[webpack-dev-server] Server started: Hot Module Replacement    index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                        core.mjs:26656
```

Click on the input boxes and then click outside the boxes to see the form validation



Registration Form

Name:

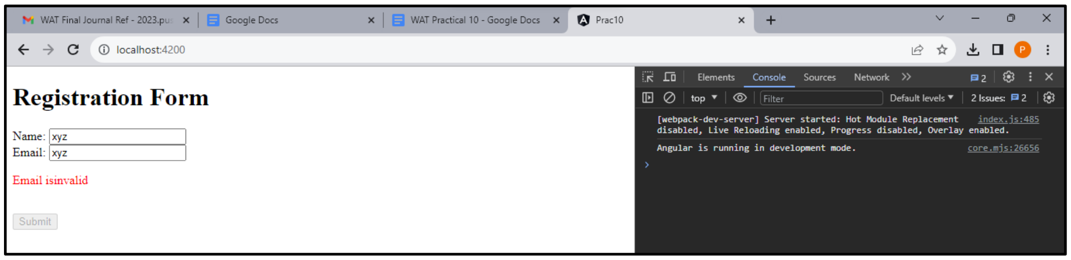
Name is invalid

Email:

Email isinvalid

```
[webpack-dev-server] Server started: Hot Module Replacement    index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                          core.mjs:26656
```

Type an invalid email to see the email's validation



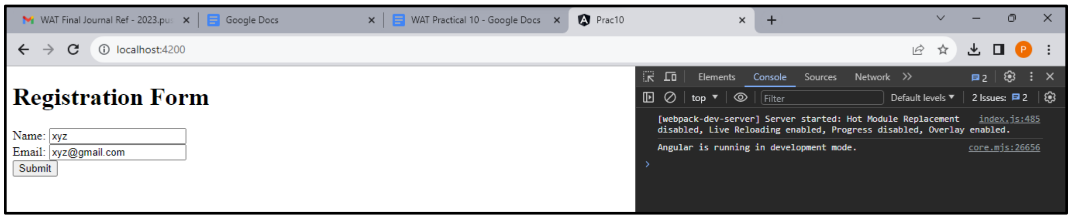
Registration Form

Name: xyz
Email: xyz

Email isinvalid

```
[webpack-dev-server] Server started: Hot Module Replacement    index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                          core.mjs:26656
```

Type all valid input values to enable the Submit Button

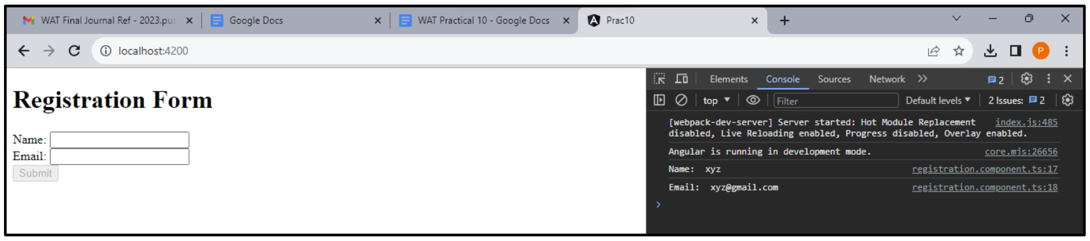


Registration Form

Name: xyz
Email: xyz@gmail.com

```
[webpack-dev-server] Server started: Hot Module Replacement    index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                          core.mjs:26656
```

Submit the form to see the values in the console



Registration Form

Name:
Email:

```
[webpack-dev-server] Server started: Hot Module Replacement    index.js:485
disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.                          core.mjs:26656
Name:  xyz                                        registration.component.ts:17
Email:  xyz@gmail.com                             registration.component.ts:18
```

Conclusion: We learnt about forms in Angular js.

Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 11**	
Title of Lab Assignment: Create an application to demonstrate SPA.			
DOP: 24-10-2023		**DOS: 24-10-2023**	
CO Mapped: **CO6**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical No. 11

Aim: Create an application to demonstrate SPA.

Theory:

1. **Single page application (SPA):**
   SPA is a web application that fits on a single page. All your code (JavaScript, HTML, and CSS) is recovered with a single page stack. Furthermore, route between pages performed without invigorating the entire page.

2. **Why is Angular called a single-page application?**
   AngularJS is a full featured SPA framework, with the help of which a single page application is created. In the SPA, the whole page is not reloaded every time, only every time the view will be changed.
   So when you load the application for the first time, not all the pages from the server will be rendered... It's only index.html that loads when you load the application. Since only a single page is loaded it is called SPA.
   The name "single-page application" comes from the fact that, in an SPA, the initial HTML page is loaded once, and after that, only the data is fetched and updated dynamically, without reloading the entire page. This approach provides a more fluid and responsive user experience, similar to that of a desktop application.

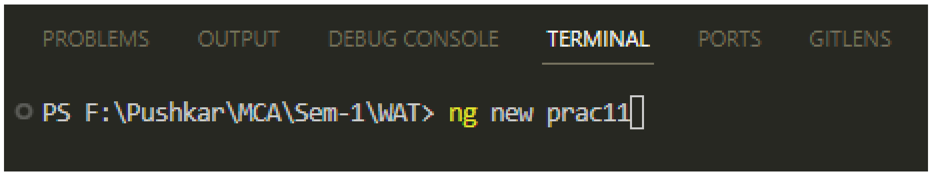
3. **Even if the URL changes in an angular site will it be called SPA?**
   In Angular applications, the URL can change, and the application is still considered a single- page application (SPA). The key characteristic of an SPA is that, even though the URL changes, the application dynamically updates and navigates within a single HTML page, without performing a full page reload.
   Angular's routing system allows you to change the URL, and when a new route is activated, it loads the appropriate component and updates the content on the page without requesting an entirely new HTML page from the server. This gives users the experience of navigating through different views or pages within the application, even though it's technically a single HTML page.

So, when you navigate to different routes in an Angular application, it's still considered a single-page application because the core HTML page remains the same, and the content changes dynamically, providing a seamless and responsive user experience.
The URL changes, because the router sets a virtual URL on your defined routes. This is just to fool the browser and make the go back and forth functionality work.

1. **Create an application to demonstrate SPA.**
   Steps to execute the practical:
   1) Create a folder "practical 11"

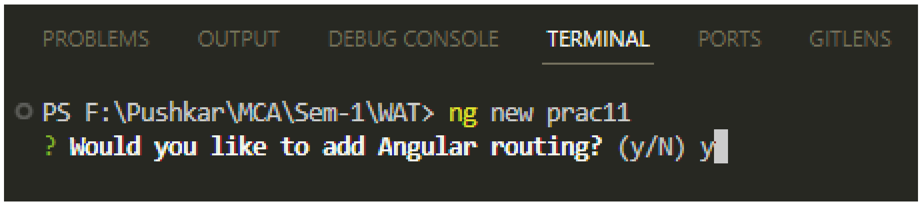
   2) Open the terminal (please use cmd, avoid using powershell) and navigate to the "practical 11" folder by using the cd command

   3) Create a new angular project by running the following command
      ng new prac11



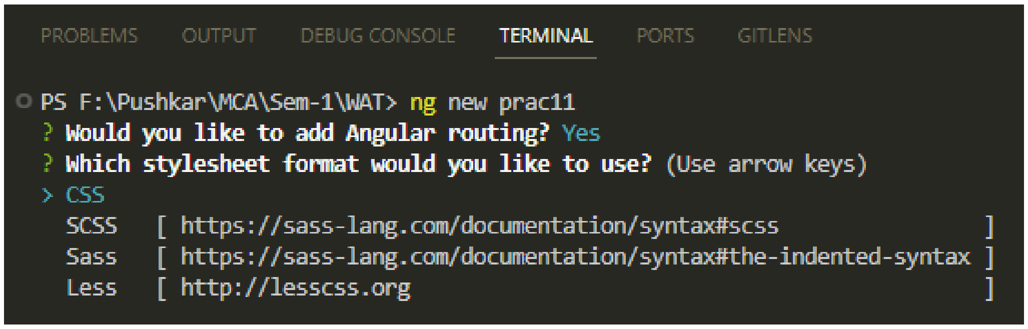
```
PS F:\Pushkar\MCA\Sem-1\WAT> ng new prac11
```

   4) When prompted for Angular Routing, Enter "y"



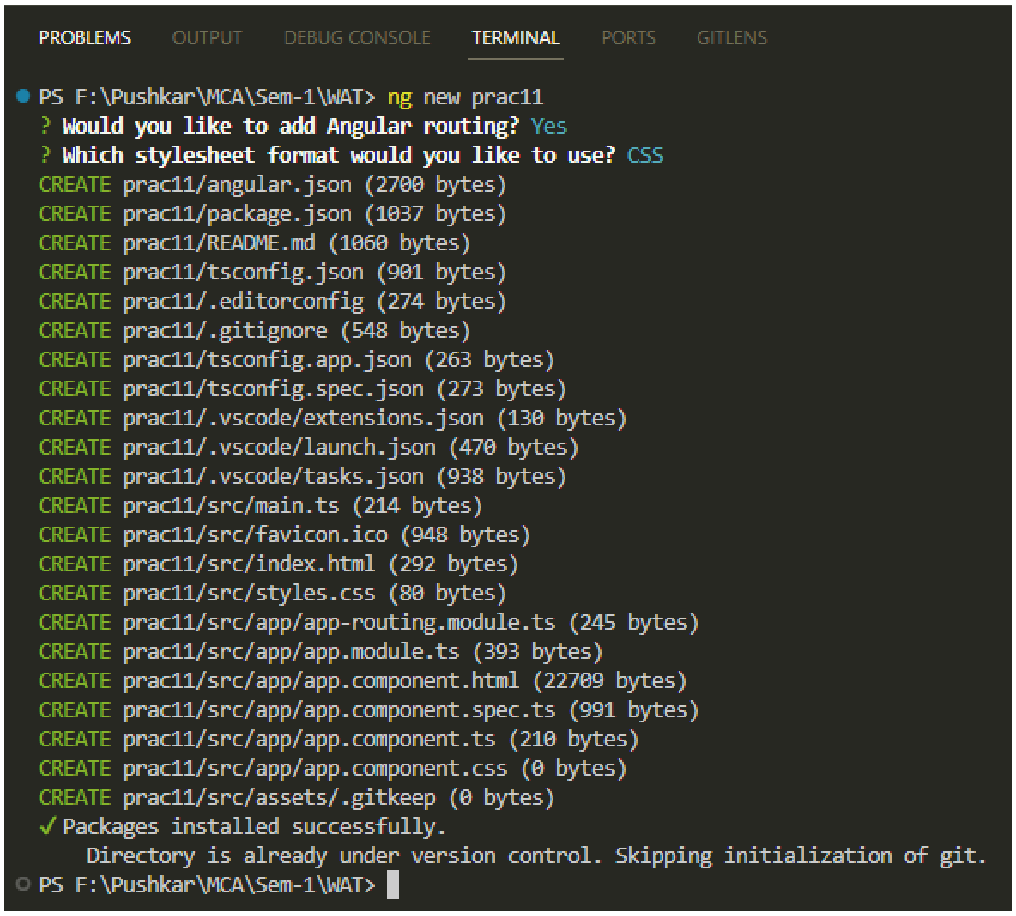
```
PS F:\Pushkar\MCA\Sem-1\WAT> ng new prac11
? Would you like to add Angular routing? (y/N) y
```

5) When prompted for CSS, use arrow keys to select “CSS” (it is the first option and is selected by default) and then press Enter



```
PS F:\Pushkar\MCA\Sem-1\WAT> ng new prac11
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
```

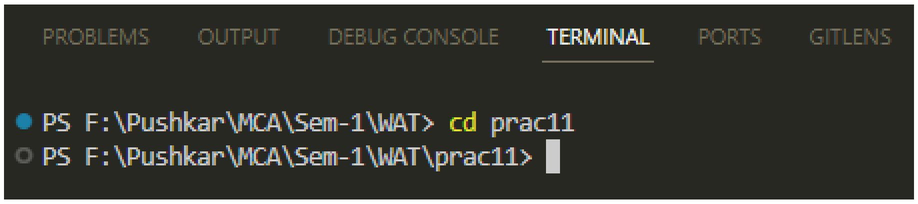
6) The new Angular project is created



```
PS F:\Pushkar\MCA\Sem-1\WAT> ng new prac11
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE prac11/angular.json (2700 bytes)
CREATE prac11/package.json (1037 bytes)
CREATE prac11/README.md (1060 bytes)
CREATE prac11/tsconfig.json (901 bytes)
CREATE prac11/.editorconfig (274 bytes)
CREATE prac11/.gitignore (548 bytes)
CREATE prac11/tsconfig.app.json (263 bytes)
CREATE prac11/tsconfig.spec.json (273 bytes)
CREATE prac11/.vscode/extensions.json (130 bytes)
CREATE prac11/.vscode/launch.json (470 bytes)
CREATE prac11/.vscode/tasks.json (938 bytes)
CREATE prac11/src/main.ts (214 bytes)
CREATE prac11/src/favicon.ico (948 bytes)
CREATE prac11/src/index.html (292 bytes)
CREATE prac11/src/styles.css (80 bytes)
CREATE prac11/src/app/app-routing.module.ts (245 bytes)
CREATE prac11/src/app/app.module.ts (393 bytes)
CREATE prac11/src/app/app.component.html (22709 bytes)
CREATE prac11/src/app/app.component.spec.ts (991 bytes)
CREATE prac11/src/app/app.component.ts (210 bytes)
CREATE prac11/src/app/app.component.css (0 bytes)
CREATE prac11/src/assets/.gitkeep (0 bytes)
✓ Packages installed successfully.
    Directory is already under version control. Skipping initialization of git.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

7) Navigate to the prac11 folder by using the cd command

cd prac11

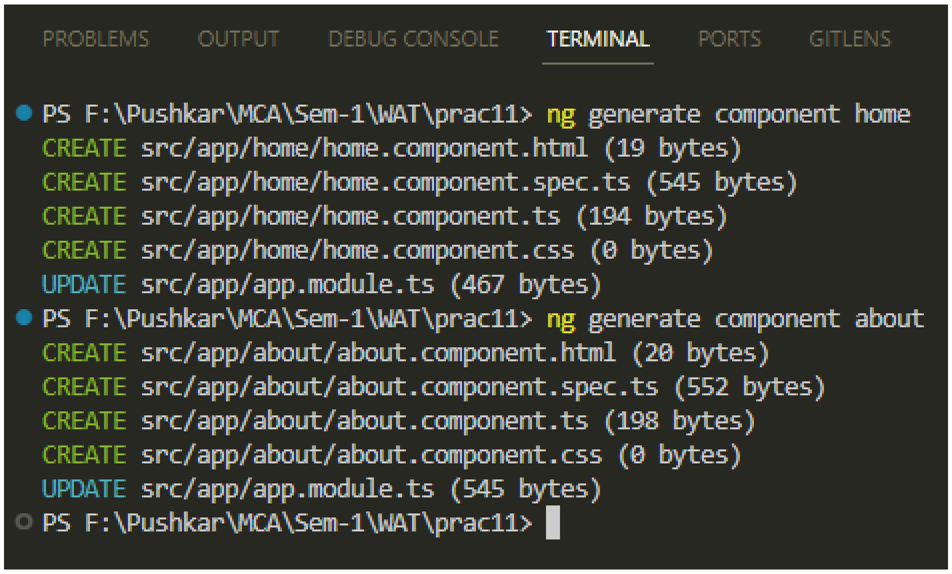


```
● PS F:\Pushkar\MCA\Sem-1\WAT> cd prac11
○ PS F:\Pushkar\MCA\Sem-1\WAT\prac11>
```

8) Create a new component "home" and "about" by running the following commands.

ng generate component home

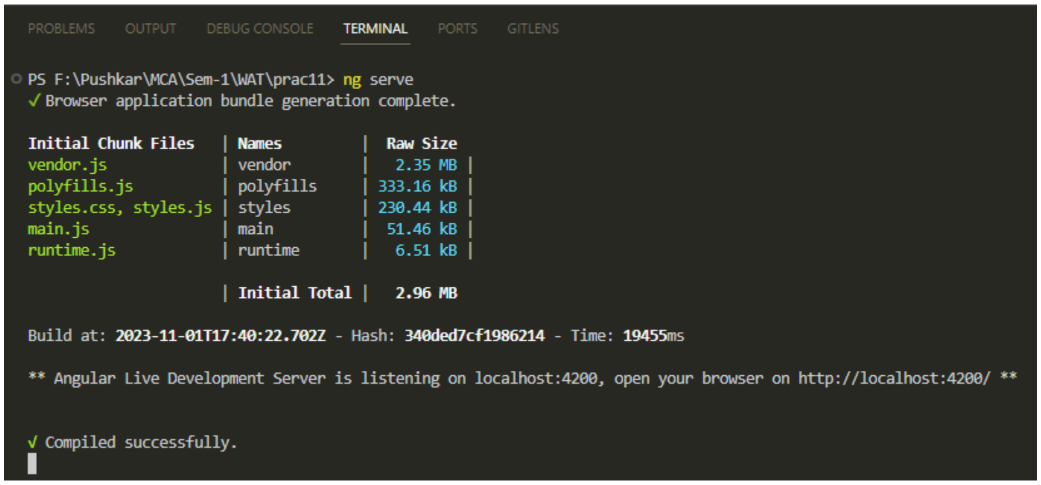
ng generate component about



```
● PS F:\Pushkar\MCA\Sem-1\WAT\prac11> ng generate component home
  CREATE src/app/home/home.component.html (19 bytes)
  CREATE src/app/home/home.component.spec.ts (545 bytes)
  CREATE src/app/home/home.component.ts (194 bytes)
  CREATE src/app/home/home.component.css (0 bytes)
  UPDATE src/app/app.module.ts (467 bytes)
● PS F:\Pushkar\MCA\Sem-1\WAT\prac11> ng generate component about
  CREATE src/app/about/about.component.html (20 bytes)
  CREATE src/app/about/about.component.spec.ts (552 bytes)
  CREATE src/app/about/about.component.ts (198 bytes)
  CREATE src/app/about/about.component.css (0 bytes)
  UPDATE src/app/app.module.ts (545 bytes)
○ PS F:\Pushkar\MCA\Sem-1\WAT\prac11>
```

9) Run the following command to serve the angular project on a server

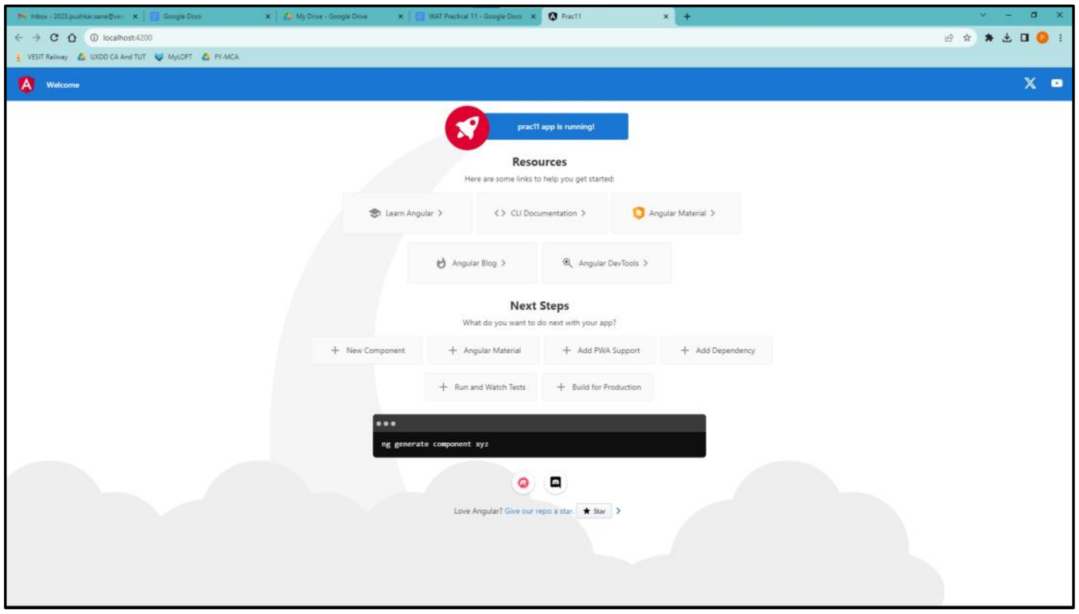
ng serve



```
PS F:\Pushkar\MCA\Sem-1\WAT\prac11> ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   2.35 MB |
polyfills.js          | polyfills     | 333.16 kB |
styles.css, styles.js | styles        | 230.44 kB |
main.js               | main          |  51.46 kB |
runtime.js            | runtime       |   6.51 kB |

                      | Initial Total |   2.96 MB

Build at: 2023-11-01T17:40:22.702Z - Hash: 340ded7cf1986214 - Time: 19455ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
```

10) The server is running and will automatically re-compile the project when we make any changes

11) Open any browser and go to the following link to see the output (this is the default output whenever any new Angular project is made) http://localhost:4200/



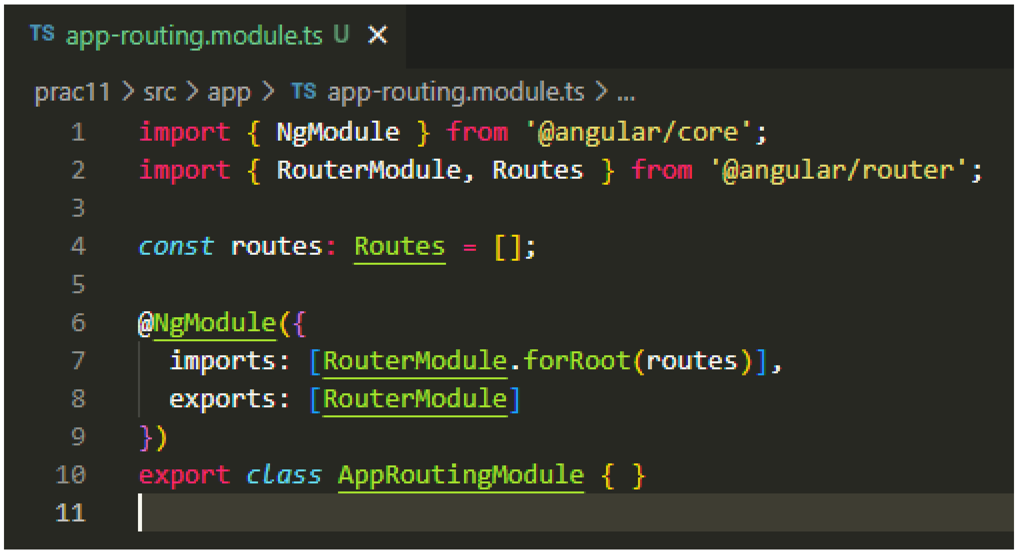
12) Add the “home” and “about” component to the **app.component.html** file

Delete all the contents of app.component.html file (located in the prac11/src/app/ folder) Add the following code to the file and save it:

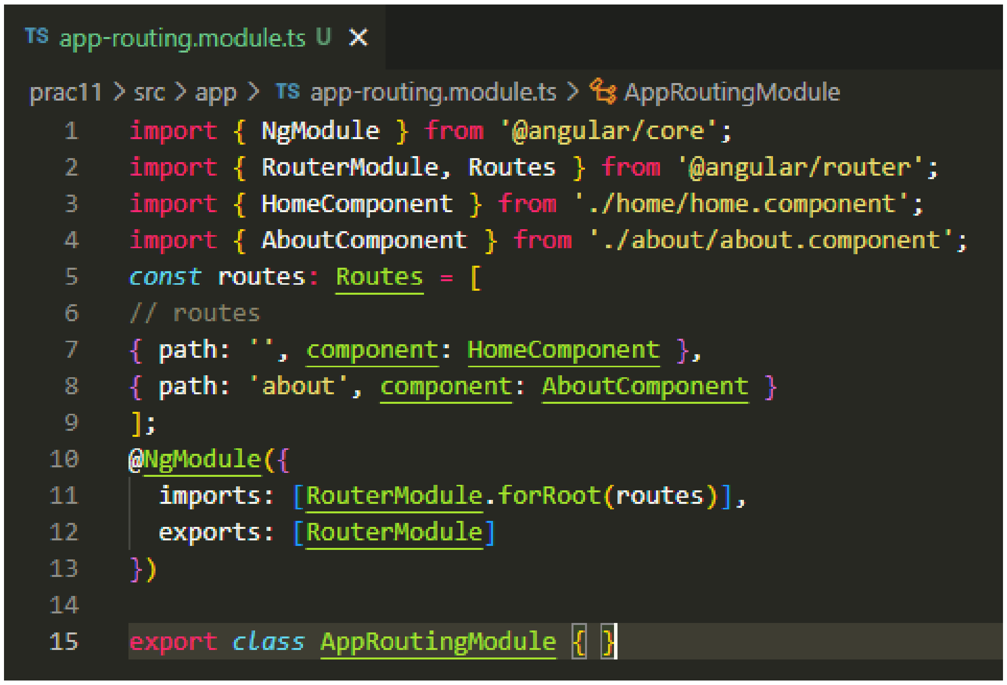
```
<nav>
  <a routerLink="/">Home Link</a>
  <br>
  <a routerLink="/about">About Link</a>
</nav>
<router-outlet></router-outlet>
```

13) We will add routing to our Angular project

To do so, we have to modify the **app-routing.module.ts** in the “app” folder (prac11/src/app/)


TS app-routing.module.ts U ×
prac11 > src > app > TS app-routing.module.ts > ...

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```


app-routing.module.ts U
prac11 > src > app > app-routing.module.ts > AppRoutingModule

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
const routes: Routes = [
// routes
{ path: '', component: HomeComponent },
{ path: 'about', component: AboutComponent }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

export class AppRoutingModule { }
```

After making the changes, the **app-routing.module.ts** file should look like this

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
const routes: Routes = [
// routes
{ path: '', component: HomeComponent },
{ path: 'about', component: AboutComponent }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

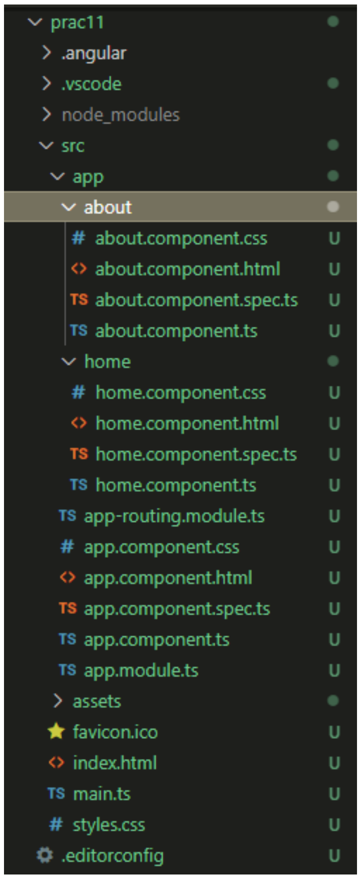
14) We will remove the default code that’s generated and add our practical code. We will make changes to the following 2 files in the “home” and “about” folders (prac11/src/app/home/

and

prac11/src/app/about/)

home.component.html

about.component.html



```
prac11
  .angular
  .vscode
  node_modules
  src
    app
      about
        about.component.css          U
        about.component.html         U
        about.component.spec.ts      U
        about.component.ts           U
      home
        home.component.css           U
        home.component.html          U
        home.component.spec.ts       U
        home.component.ts            U
      app-routing.module.ts          U
      app.component.css              U
      app.component.html             U
      app.component.spec.ts          U
      app.component.ts               U
      app.module.ts                  U
    assets
    favicon.ico                      U
    index.html                       U
    main.ts                          U
    styles.css                       U
  .editorconfig                      U
```

15) Open the **home.component.html** file and delete all the code

16) Write the following code in **home.component.html**

    <p>home page</p>

17) Save the **home.component.html** file.

18) Open the **about.component.html** file and delete all the code

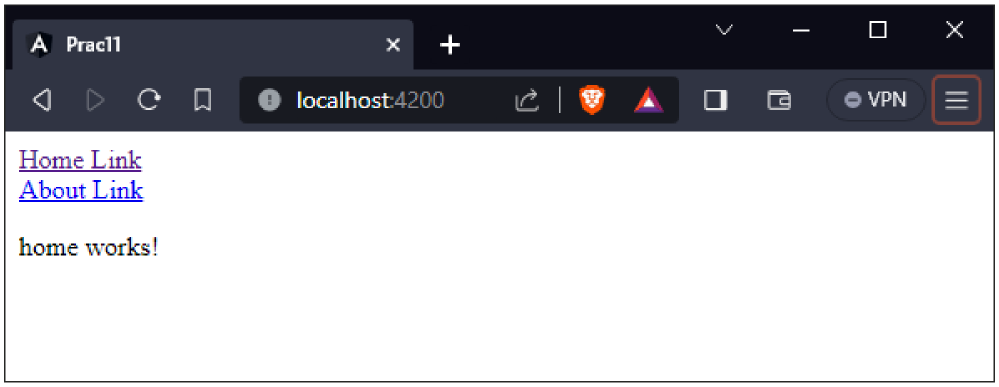
19) Write the following code in about.component.html

    <p>about page</p>

20) Save the **about.component.html** file

21) Go to the browser to see the output at http://localhost:4200/

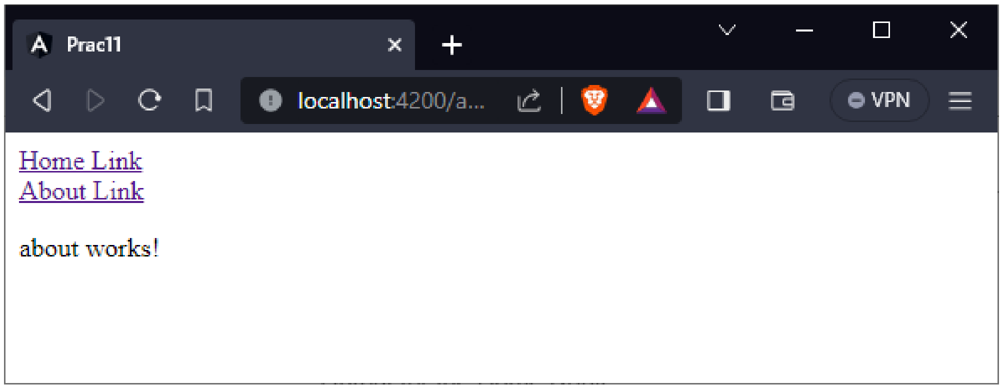
Output for the Home Route



Home Link
About Link

home works!

Output for the About Route

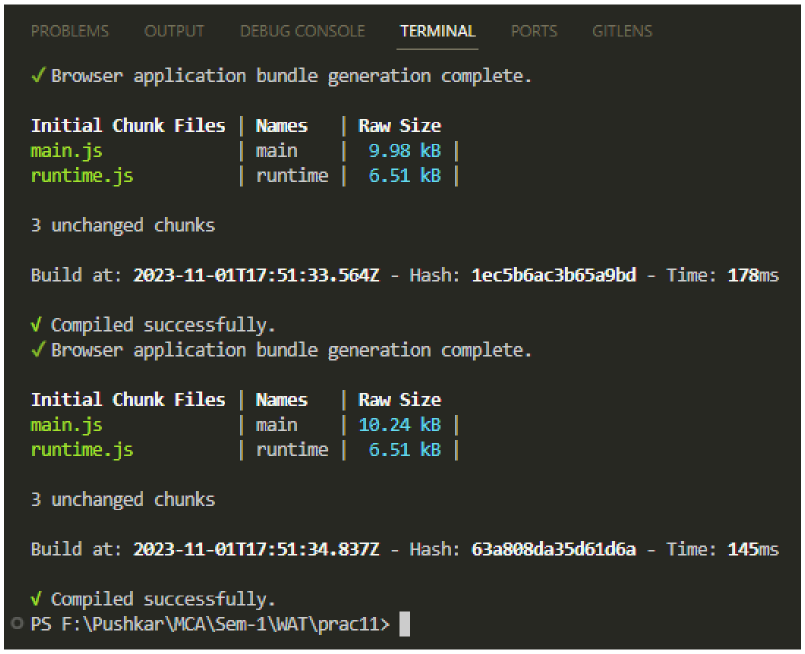


Home Link
About Link

about works!

22) Go to the terminal and stop the server by pressing Ctrl + C

When we see the prompt as "....\prac11>" then it means that the server has stopped



```
✓ Browser application bundle generation complete.

Initial Chunk Files | Names   | Raw Size
main.js             | main    |  9.98 kB |
runtime.js          | runtime |  6.51 kB |

3 unchanged chunks

Build at: 2023-11-01T17:51:33.564Z - Hash: 1ec5b6ac3b65a9bd - Time: 178ms

√ Compiled successfully.
✓ Browser application bundle generation complete.

Initial Chunk Files | Names   | Raw Size
main.js             | main    | 10.24 kB |
runtime.js          | runtime |  6.51 kB |

3 unchanged chunks

Build at: 2023-11-01T17:51:34.837Z - Hash: 63a808da35d61d6a - Time: 145ms

√ Compiled successfully.
PS F:\Pushkar\MCA\Sem-1\WAT\prac11>
```

<u>Code:</u>

app.component.html

```
<nav>
  <a routerLink="/">Home Link</a>
  <br>
  <a routerLink="/about">About Link</a>
</nav>
<router-outlet></router-outlet>
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
const routes: Routes = [
// routes
{ path: '', component: HomeComponent },
{ path: 'about', component: AboutComponent }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

export class AppRoutingModule { }
```

home.component.html

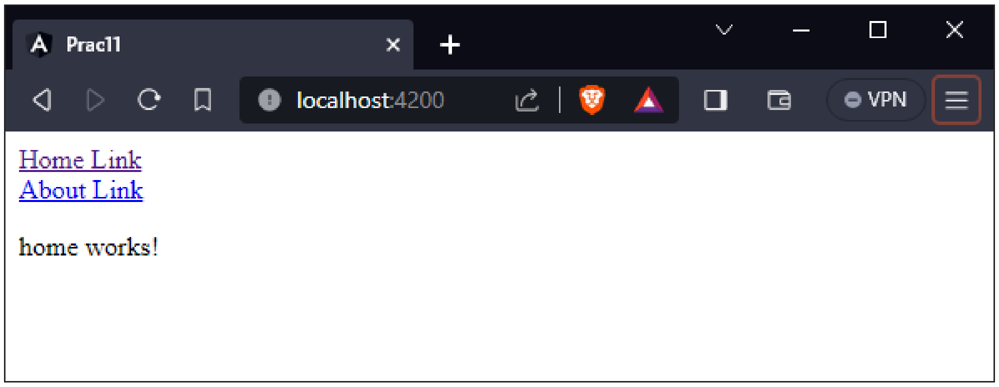
```
<p>home works!</p>
```

about.component.html

```
<p>about works!</p>
```

Output:

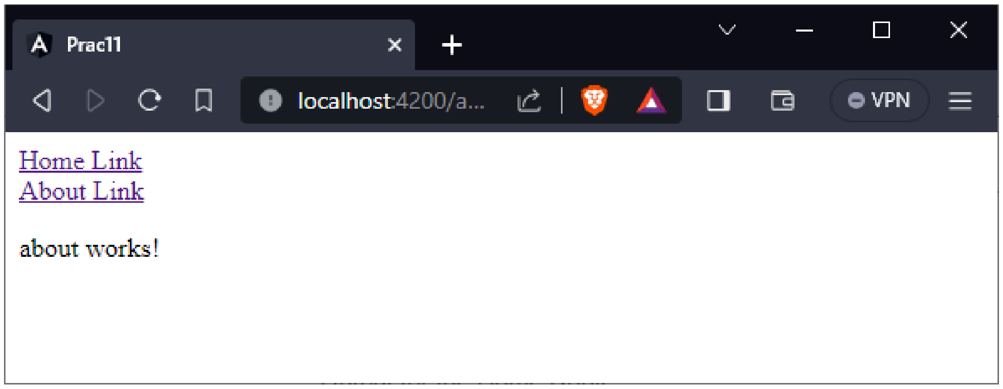
Home route



Home Link
About Link

home works!

About route



Home Link
About Link

about works!

Conclusion: We learnt about SPA in Angular js.