**AIM:File Operations Create a text file src.txt and add the following data to it. Mongo, Express, Angular, Node.**

**Description:**

The Node.js file system module allows you to work with the file system on your computer.

To include the File System module, use the require() method:

var fs = require('fs');

Common use for the File System module:

Read files Create files Update files Delete files Rename files

The File System module has the following methods for creating a new file and writing data to that file:

writeFile()

appendFile()

**WriteFile:**

The fs.writeFile() method will overwrite the content if the content already exists.

If the file does not exist, then a new file will be created with the specified name and content.

Syntax:

fs.writeFile(file, data, callback);

**file:** Placeholder to give the file name in which you are going to write the data.

data: The data/content must be written to the file.

callback: The callback method, that will be executed, when 'writeFile()' function is executed. This callback will be executed in both success as well as failure scenarios.

**AppendFile:**

The appendFile() method first checks if the file exists or not. If the file does not exist, then it creates a new file with the content, else it appends the given content to the existing file.

**Syntax:**fs.appendFile(path, data, callback)

path: Placeholder to give the file name in which you are going to append the data.

data: The data/content which must be appended to the file.

callback: The callback method, that will be executed, when 'appendFile()' function is executed. This callback will be executed in both success as well as failure scenarios.

**Program:**

var fs = require('fs');

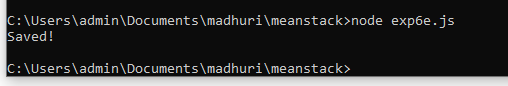
fs.appendFile('src.txt','Mongo , Express ,Angular , Node ', function (err) {

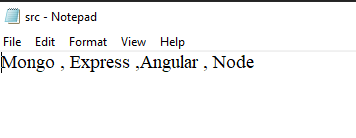
if (err) throw err;

console.log('Saved!');

});

Output:





**AIM: Course Name: Node.js Module Name: Restarting Node Application Write a program to show the workflow of restarting a Node application**

**Description:**

Whenever we are working on a Node.js application and we do any change in code after the application is started, we will be required to restart the Node process for changes to reflect. In order to restart the server and to watch for any code changes automatically, we can use the Nodemon tool.

Nodemon

Nodemon is a command-line utility that can be executed from the terminal. It provides a different way to start a Node.js application. It watches the application and whenever any change is detected, it restarts the application.

It is very easy to get started with this tool. To install it in the application, run the below command.

npm install nodemon –g

Once the 'nodemon' is installed in the machine, the Node.js server code can be executed by replacing the command "node" with "nodemon".

**Program:**

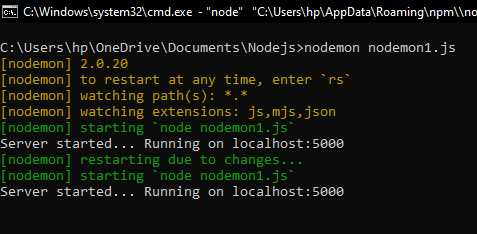
const http = require("http");

var server = http.createServer((req, res) => { res.writeHead(200,{'Content-Type': 'text/html'}); res.write("Hello ! I have created my second server!"); res.end();

});

server.listen(5000);

console.log("Server started... Running on localhost:5000");

**Output:**



**AIM: Introduction to Asynchronous Programming, Callbacks, Promises, Async and Await, Executing Network Requests using Fetch API Simulate a periodic stock price change and display on the console. Hints: (i) Create a method which returns a random number - use Math.random, floor and other methods to return a rounded value. (ii) Invoke the method for every three seconds and stop when the count is 5 – use the setInterval method. (iii) Since setInterval is an async method, enclose the code in a Promise and handle the response generated in a success callback. (iv) The random value returned from the method every time can be used as a stock price and displayed on the console.**

**Aim:**To stimulate a periodic stock price change and display on the console.

**Description:**

To use the random function

**Syntax: Math.random()**

To use the setInterval function

**Syntax:myInterval = setInterval(function, milliseconds);**

To stop the execution of setInterval function

**Syntax:clearInterval(myInterval);**

To create a Promise we have to use to following Syntax

**Syntax:**

**let myPromise = new Promise(function(Resolve, Reject) {  
  
  Resolve(); // when successful  
  Reject();  // when error  
});**

**Program:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Exp\_\_5b</title>

</head>

<body>

<script>

let c=0;

const stock=setInterval(stokc,3000);

function stokc(){

var myPromise = new Promise(function (resolve, reject)

{

setTimeout(function () {

var a=Math.floor(Math.random() \* 10);

resolve(a);

}, 3000);

});

myPromise.then(

function (data) {

console.log(data);

},

function (error) {

console.log(error);

}

);

c+=1;

if(c==5)

{

Stop();

}

}

function Stop() {

clearInterval(stock);

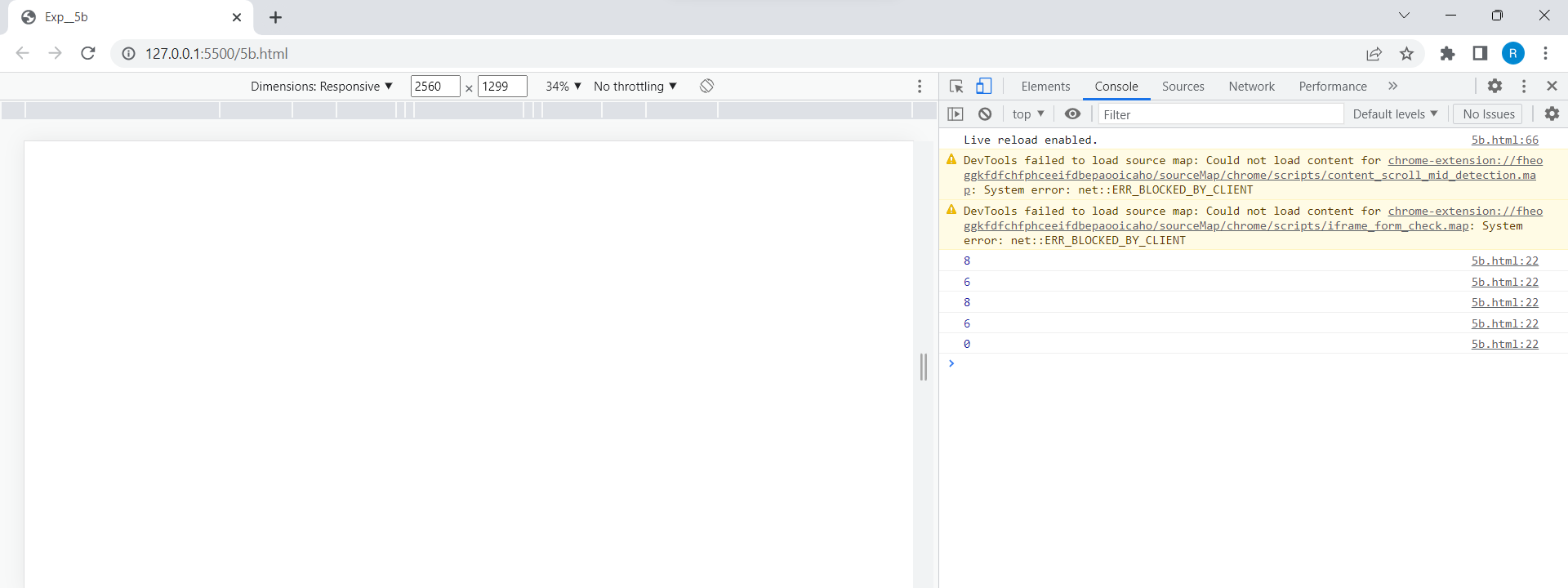
}

</script>

</body>

</html>

**Output:**

****