

Deccan Education Society's
Navinchandra Mehta Institute of
Technology and Development
C E R T I F I C A T E

This is to certify that Mr. / Miss. **Nishant D. Desai** of M.C.A. Semester I with Roll No. **C23029** has completed _____ practicals of Web Technologies under my supervision in this college during the year 2022-2023.

CO	R1 (Journal)	R2 (Performance during lab session)	R3 (Implementation using different problem solving techniques)	R4 (Mock Viva)	Attendance
CO1					
CO2					
CO3					
CO4					

Practical-in-charge

Head of Department
MCA Department
(NMITD)

MCAL14 Web Technology Lab Index

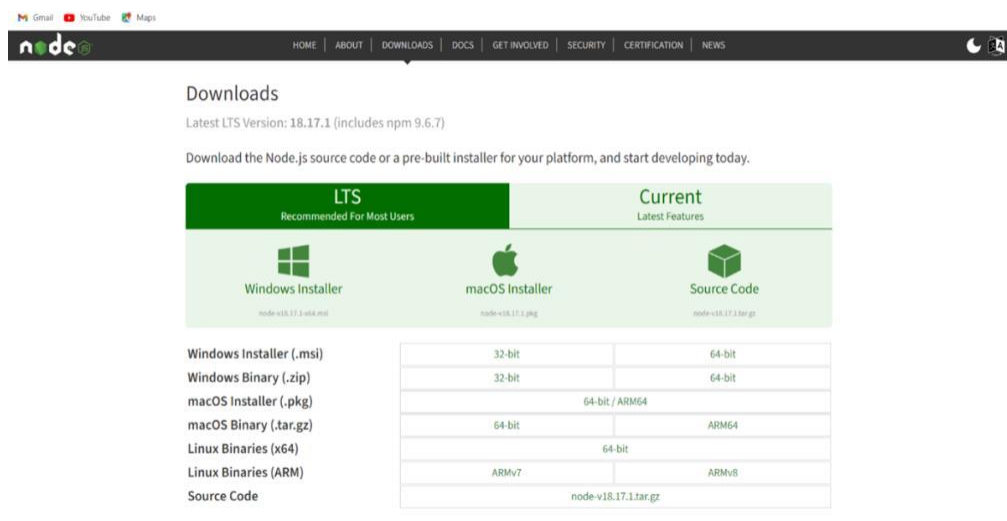
Sr.No	Topic Name	Date	CO	Sign
1	Create an application to demonstrate Node.js Modules		CO1	
2	Create an application to demonstrate various Node.js Events		CO1	
3	Create an application to demonstrate Node.js Functions		CO1	
4	Using File Handling demonstrate all basic file operations (Create, write, read, delete)		CO1	
5	Create an HTTP Server and perform operations on it		CO1	
6	Create an application to establish a connection with the MySQL database and perform basic database operations on it		CO2	
7	Create an application using Filters		CO3	
8	Create an application to demonstrate directives		CO3	
9	Demonstrate controllers in Angular.js through an application		CO3	
10	Demonstrate features of Angular.js forms with a program		CO3 CO4	
11	Create a SPA (Single Page Application)		CO4	

Web Technology journal

Nishant Desai (C-23029)

Steps to Install Node.js Application:-

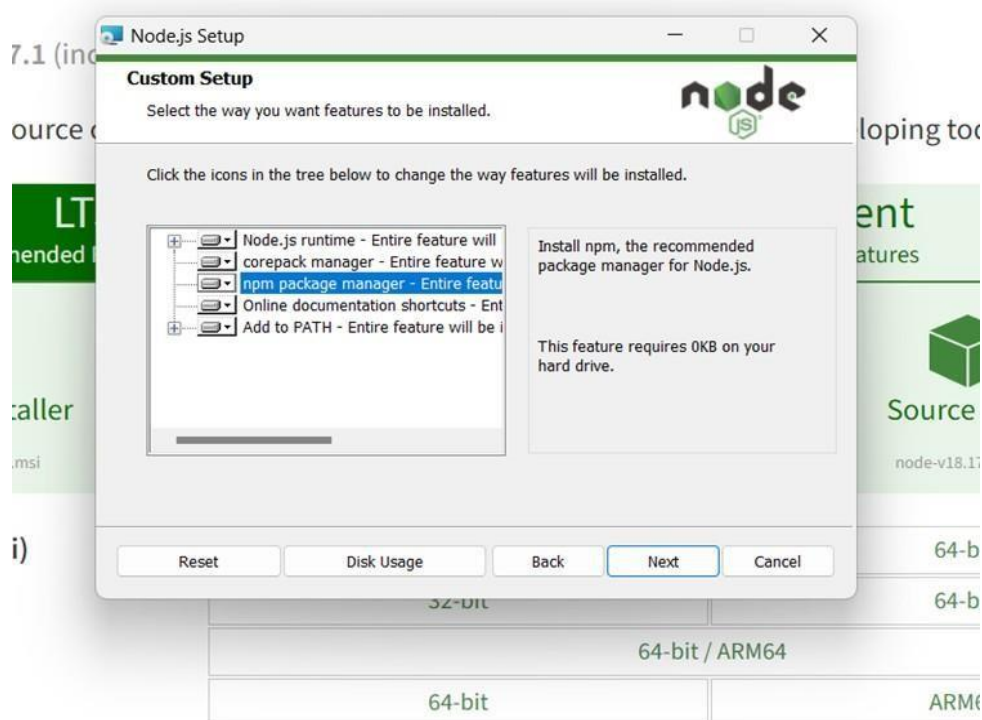
1. Download Node.JS Installer for Windows.
2. Open a Web browser.
3. Go to <https://nodejs.org/en/download>.
4. Download node MSI for windows by clicking on 8.11.3 LTS or 10.5.0 Current button. <https://nodejs.org/en/download/>



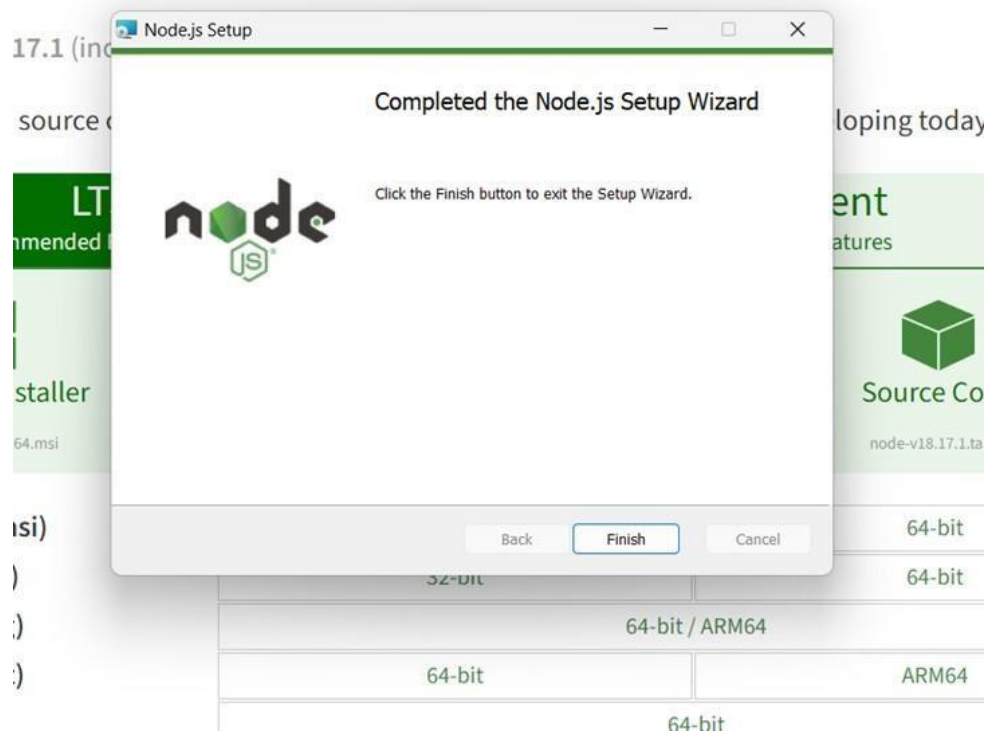
Step 2:- Click Next to read and accept the License Agreement and then click Install. It will install Node.js quickly on your computer



Step 3:- Install Node.js and npm package.



Step 4:- Click the Finish Button To Complete The Installation part.



Step 5:- After the installation is complete, you can verify that Node.js and npm (Node Package Manager) are installed correctly by opening a terminal or command prompt and running the following commands.

```
Command Prompt
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nishd>node -v
v16.13.0

C:\Users\nishd>npm -v
8.1.0

C:\Users\nishd>
```

1) Create an application to demonstrate Node.js Modules

Modules

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

Include Modules :-

To include a module, use the `require()` function with the name of the module: `Const calculator = require('./calculator');`

Example :-

Step 1

Create the Calculator Module (calculator.js): Inside the calculator.js file, define the module: This module provides basic calculator function

```
function add(a, b) {  
  return a + b;  
}
```

```
// Function to subtract two numbers  
function subtract(a, b) {  
  return a - b;  
}
```

```
// Function to multiply two numbers  
function multiply(a, b) {  
  return a * b;  
}
```

```
// Function to divide two numbers  
function divide(a, b) {  
  if (b === 0) {
```

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

Export the calculator functions as a module

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

Step 2

Use the Calculator Module in the Main Application :

Inside the app.js file, require the calculator module and use its functions

Require the calculator module

```
const calculator = require("./calculator");
const num1 = 100;
const num2 = 200;
```

```
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)}`);
try {
  console.log(`Division: ${num1} / ${num2} = ${calculator.divide(num1, num2)}`);
} catch (error) {
  console.error(error.message);
}
```

Output:-


```
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\nishd\Desktop\Assingment1> node module.js
Addition: 100 + 200 = 300
Subtraction: 100 - 200 =
-100
Multiplication: 100 *
200 = 20000
Division: 100 / 200 = 0.5
PS C:\Users\nishd\Desktop\Assingment1>
```

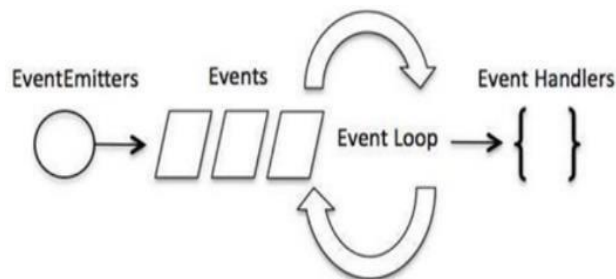
2) Create an application to demonstrate Node.js Events

Node.js Events

In Node.js applications, Events and Callbacks concepts are used to provide concurrency. As Node.js applications are single threaded and every API of Node.js are asynchronous. So it uses async function to maintain the concurrency. Node uses observer pattern. Node thread keeps an event loop and after the completion of any task, it fires the corresponding event which signals the event listener function to get executed.

Event Driven Programming

Node.js uses event driven programming. It means as soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for event to occur. It is the one of the reason why Node.js is pretty fast compared to other similar technologies. There is a main loop in the event driven application that listens for events, and then triggers a callback function when one of those events is detected.



Node.js Event Example

Step 1

Initialize a Node.js Project: Open your terminal and create a new directory for your project. Navigate to that directory and run the following commands to initialize a Node.js project and install the necessary dependencies

1. mkdir nodejs-events-demo
2. cd nodejs-events-demo
3. npm init -y
4. npm install events

Step 2

Creating a JavaScript file by any name in my project directory.

Step 3

Write the Node.js Application // Import the events module

```
const EventEmitter = require("events");
```

```
// Create an instance of EventEmitter
```

```
const emitter = new EventEmitter();
```

```
// Define event listeners
```

```
emitter.on("customEvent", (message) => {  
  console.log(`Listener 1: ${message}`);  
});
```

```
emitter.on("customEvent", (message) => {  
  console.log(`Listener 2: ${message}`);  
});
```

```
// Emit the custom event
```

```
emitter.emit("customEvent", "Heyy Nishant");
```

```
// Remove one of the event listeners
```

```
emitter.removeListener("customEvent", (message) => {  
  console.log(`Listener 2 removed: ${message}`);  
});
```

```
// Example of using once to listen for an event only once emitter.once('onceEvent', () => {  
  console.log("This event will only trigger once.");
```

```
emitter.emit("onceEvent");
```

```
emitter.emit("onceEvent"); // This won't trigger the listener again
```

```
// Example of using the listenerCount method
```

```
const listenerCount = EventEmitter.listenerCount(emitter, "customEvent");  
console.log(`Total customEvent listeners: ${listenerCount}`);
```

Output:-



```
PS C:\Users\nishd\Desktop\Assignment1> node eventEmitter.js  
Listener 1: Heyy Nishant  
Listener 2: Heyy Nishant  
This event will only trigger once.  
Total customEvent listeners: 2  
PS C:\Users\nishd\Desktop\Assignment1>
```

3) Create an application to demonstrate Node.js Functions.

Functions in Node.js

i) A function is the core of NodeJS, NodeJS uses functions everywhere such as for standard operations, as an arrow function, or as a callback function.

ii) The function is the key fundamental of programming languages, it is used to combine different expressions and instruct them to perform certain operations.

iii) A function makes the overall structure of the program better by providing a feature to group certain expressions and operations, which also helps in reducing duplicate code.

How to Define a Function in NodeJS?

Function Definition means defining the function behaviour, functionality, name, parameters, and the value it returns. In other words, before using a function it is required to create that function, and creating a function is called the function definition.

Step 1

create a basic Node.js application that calculates the factorial of a number using a custom function.

```
// Function to calculate the factorial of a number
function factorial(num)
{
  if (num === 0 || num === 1) { return 1;
}
  else {
    return num * factorial(num - 1);
  }
}
// Example usage of the factorial function
const number = 20;
const result = factorial(number);
console.log(`The factorial of ${number} is ${result}`);
```

Output :-



A screenshot of a Visual Studio Code terminal window. The terminal has a dark background with light blue and yellow text. At the top, there is a tab bar with 'PROBLEMS' (with a purple icon), 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'COMMENTS'. The 'TERMINAL' tab is active. To the right of the tab bar is a search bar with the placeholder text 'Filter (e.g. text, exclude)' and a close button. The terminal content shows a command prompt where the command `C:\Program Files\nodejs\node.exe .\factorial.js` has been executed. The output is `The factorial of 20 is 2432902008176640000`. On the right side of the terminal, the text `factorial.js:12` is visible, indicating the current line of code being executed.

```
C:\Program Files\nodejs\node.exe .\factorial.js
The factorial of 20 is 2432902008176640000
factorial.js:12
```

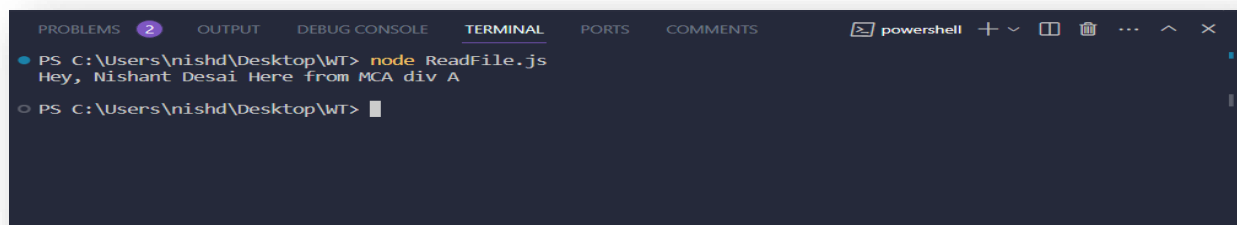
4) Using file Handling demonstrate all basic file operations (Create, Write,read,delete)

ReadFile.js:

Code:

```
var fs = require("fs");
fs.readFile("input.txt", function (err, data) {
  if (err) throw err;
  console.log(data.toString());
});
```

Output:

A screenshot of a PowerShell terminal window. The window has a dark background and a title bar that says "powershell". The terminal shows the command "node ReadFile.js" being executed. The output is "Hey, Nishant Desai Here from MCA div A". The prompt "PS C:\Users\nishd\Desktop\WT>" is visible at the bottom.

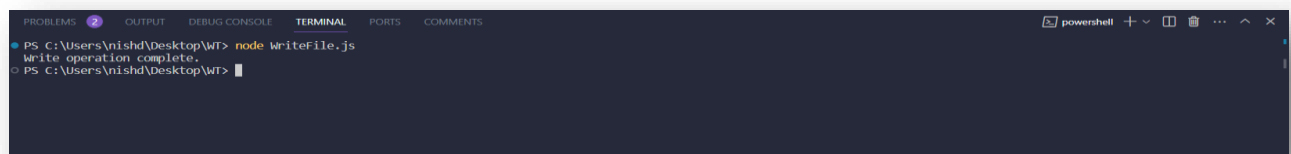
```
PS C:\Users\nishd\Desktop\WT> node ReadFile.js
Hey, Nishant Desai Here from MCA div A
PS C:\Users\nishd\Desktop\WT>
```

WriteFlie.js

Code:

```
var fs = require("fs");
fs.writeFile("test.txt", "Hello World hi 123!", function (err) {
  if (err) console.log(err);
  else console.log("Write operation complete.");
});
```

Output:

A screenshot of a PowerShell terminal window. The window has a dark background and a title bar that says "powershell". The terminal shows the command "node WriteFile.js" being executed. The output is "Write operation complete.". The prompt "PS C:\Users\nishd\Desktop\WT>" is visible at the bottom.

```
PS C:\Users\nishd\Desktop\WT> node WriteFile.js
Write operation complete.
PS C:\Users\nishd\Desktop\WT>
```

AppendFile.js

Code:

```
var fs = require("fs");
fs.appendFile("input.txt", "hiii ", function (err) {
  if (err) throw err;
  console.log("Saved!");
});
```

Output:

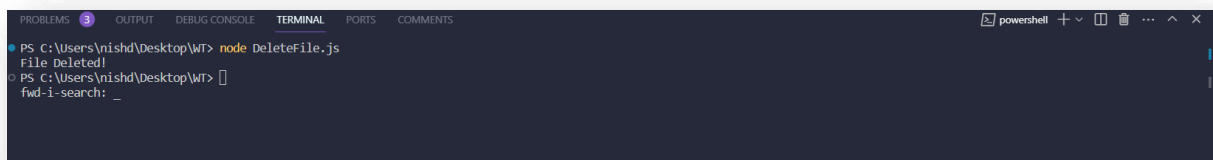
A screenshot of a terminal window with a dark background. The terminal shows the command 'node AppendFile.js' being executed. The output is 'Saved!'. The terminal window has tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'COMMENTS'. The 'TERMINAL' tab is active. The window title is 'powershell'.

DeleteFile.js

Code:

```
var fs = require("fs");
fs.unlink("input.txt", function () {
  console.log("File Deleted!");
});
```

Output:

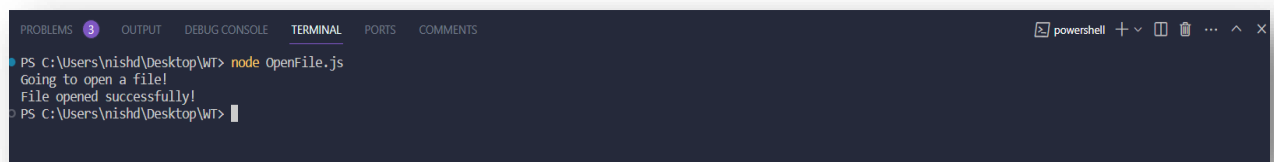
A screenshot of a terminal window with a dark background. The terminal shows the command 'node DeleteFile.js' being executed. The output is 'File Deleted!'. The terminal window has tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'COMMENTS'. The 'TERMINAL' tab is active. The window title is 'powershell'.

OpenFile.js

Code:

```
var fs=require('fs');
console.log("Going to open a
file!");
fs.open('input.txt','r+',function(err
,fd){if (err){return
console.error(err);}
console.log("File opened successfully!"); });
```

Output:

A screenshot of a PowerShell terminal window. The title bar shows 'powershell' and standard window controls. The terminal has tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'COMMENTS'. The 'TERMINAL' tab is active. The command prompt shows 'PS C:\Users\nishd\Desktop\WT> node OpenFile.js'. The output of the command is displayed in blue text: 'Going to open a file!', 'File opened successfully!', and 'PS C:\Users\nishd\Desktop\WT>'.

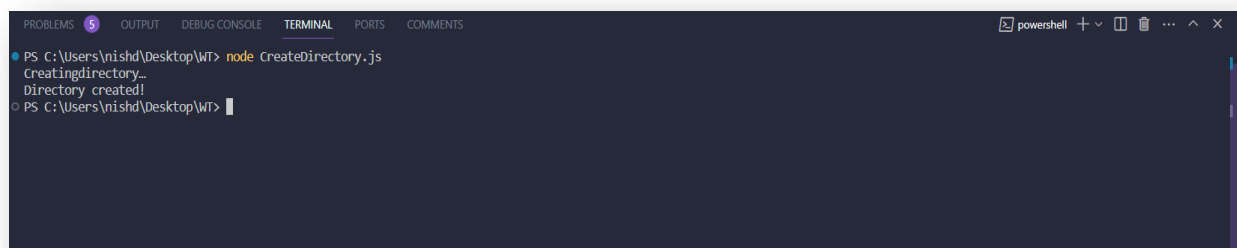
```
PS C:\Users\nishd\Desktop\WT> node OpenFile.js
Going to open a file!
File opened successfully!
PS C:\Users\nishd\Desktop\WT>
```

CreateDirectory.js

Code:

```
var fs=require('fs');
console.log("Creatingdirectory...."
);
fs.mkdir('./newdir',function(e
rr){if(err){
return console.error(err);
}
console.log("Directory created!");
});
```

Output:

A screenshot of a PowerShell terminal window. The title bar shows 'powershell' and standard window controls. The terminal has tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'COMMENTS'. The 'TERMINAL' tab is active. The command prompt shows 'PS C:\Users\nishd\Desktop\WT> node CreateDirectory.js'. The output of the command is displayed in blue text: 'Creatingdirectory...', 'Directory created!', and 'PS C:\Users\nishd\Desktop\WT>'.

```
PS C:\Users\nishd\Desktop\WT> node CreateDirectory.js
Creatingdirectory...
Directory created!
PS C:\Users\nishd\Desktop\WT>
```


ReadDirectory.js

Code:

```
var fs=require('fs');
console.log("Going to read a directory.    ");
fs.readdir('/Users',function(err,files){
    if(err){
        return console.error(err);
    }
    files.forEach(function(file
    ){console.log(file);
    });
});
```

Output:

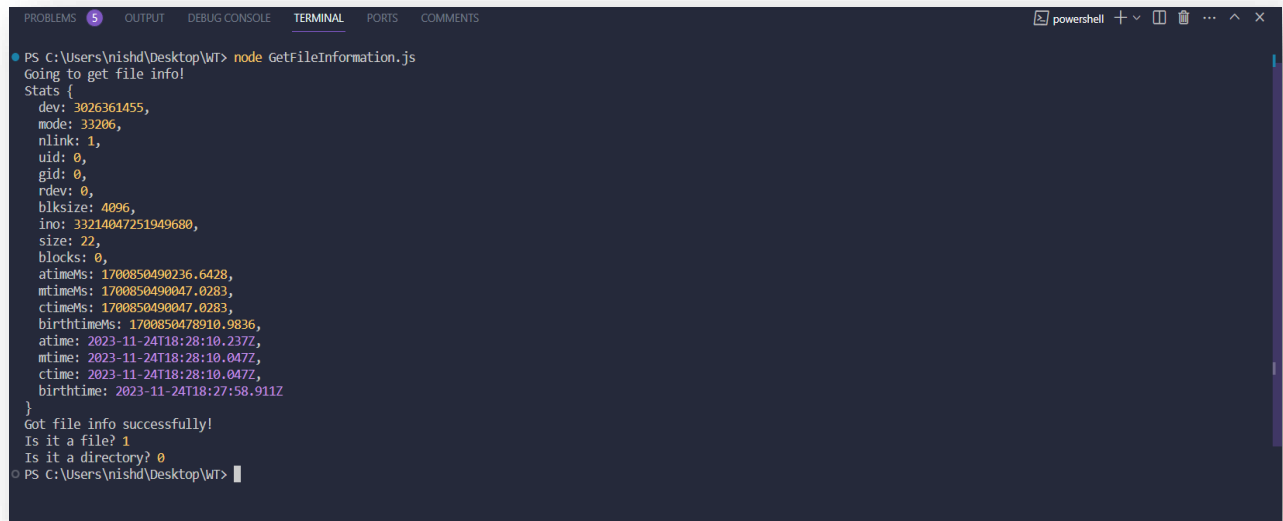
A screenshot of a PowerShell terminal window. The title bar shows 'powershell' with standard window controls. The terminal content shows the command 'node ReadDirectory.js' being executed. The output is as follows:
PS C:\Users\nishd\Desktop\WT> node ReadDirectory.js
Going to read a directory.
All Users
Default
Default User
defaultuser100000
desktop.ini
nishant
nishd
Public
PS C:\Users\nishd\Desktop\WT> |

GetFileInformation.js

Code:

```
var fs=require('fs');
console.log("Going to get file
info!");
fs.stat('input.txt','r+',function(err,st
ats){if (err){
    return console.error(err);
}
console.log(stats);
console.log("Got file info successfully!");
console.log("Is it a file?",+stats.isFile());
console.log("Is it a
directory?",+stats.isDirectory());
});
```

Output:



The screenshot shows a PowerShell terminal window within a VS Code editor. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (active), PORTS, and COMMENTS. The active terminal shows the execution of a Node.js script named 'GetFileInformation.js'. The script outputs file statistics for a file named '1'. The output is a JSON object with various file system properties. The terminal prompt is 'PS C:\Users\nishd\Desktop\WT>'.

```
PS C:\Users\nishd\Desktop\WT> node GetFileInformation.js
Going to get file info!
Stats {
  dev: 3026361455,
  mode: 33206,
  nlink: 1,
  uid: 0,
  gid: 0,
  rdev: 0,
  blksize: 4096,
  ino: 33214047251949680,
  size: 22,
  blocks: 0,
  atimeMs: 17008504900236.6428,
  mtimeMs: 17008504900047.0283,
  ctimeMs: 17008504900047.0283,
  birthtimeMs: 1700850478910.9836,
  atime: 2023-11-24T18:28:10.237Z,
  mtime: 2023-11-24T18:28:10.047Z,
  ctime: 2023-11-24T18:28:10.047Z,
  birthtime: 2023-11-24T18:27:58.911Z
}
Got file info successfully!
Is it a file? 1
Is it a directory? 0
PS C:\Users\nishd\Desktop\WT>
```

5)Create an HTTP Server and perform operations on it Node.jsWeb Server

In this section, we will learn how to create a simple Node.js web server and handle HTTP requests. To access web pages of any web application, you need a web server. The web server will handle all the http requests for the web application e.g IIS is a web server for ASP.NET web

applications and Apache is a web server for PHP or Java web applications. Node.js provides capabilities to create your own web server

which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but it is recommended to use Node.js web server.

Handle HTTP Request

The `http.createServer()` method includes request and response parameters which is supplied by Node.js. The request object can be used to get information about the current HTTP request e.g., url, request header, and data. The response object can be used to send a response for a current HTTP request. The following example demonstrates handling HTTP request and response in Node.js.

```
var http = require("http");
var server =
  http.createServer(function(req,res){if
    (req.url== '/'){

      res.writeHead(200,{ 'Content-
        Type': 'text/html' }); res.write("hey
        server .....!");
      res.end();
    }
    else if(req.url== '/student'){

      res.writeHead(200,{ 'Content-Type': 'text/html' });
      res.write('<html><body><p><style>p{color:red;}</style>This
        is student page</p></body></html>');
      res.end();
    }
  })
```

```

e      dmin'){
l
s      res.writeHead(200,{ 'Content-Type':'text/html'});
e      res.write('<html><body><h1><style> h1{background:gray;
i      color:red;}</style>This is admin page</h></body></html>');
f      res.end();
(
}r
else if (req.url == "/teacher") {
q      res.writeHead(200, { "Content-Type": "text/html" });
.      res.write(
u      "<html><body><h1><style> h1{background:gray; color:red;}</style> This is teacher
page</h></body></html>"
|      );
else
res.end('Invalid Request');
server.listen(8080);
console.log('server is running..');
a
});

```

Output:-



For Windows users, point your browser to <http://localhost:8080> and see the following result.



point your browser to <http://localhost:8080/student> and see the following result.



point your browser to <http://localhost:8080/admin> and see the following result.



point your browser to <http://localhost:8080/teacher> and see the following result.



6) Create an application to establish a connection with the MySQL database and perform basic database operations on it

Create database:

```
var mysql=require('mysql');
var
    con=mysql.createConnection
    ({host:'localhost',
    user:'root',
    password:'',
    database:'DB01'
});
con.connect(function(err)
    { if(err) throw err;
    console.log("Conected!
    ");
    con.query('CREATE DATABASE
    DB01',function(err,result){ if (err) throw err;
        console.log('Database created successfully!');
    });
});
```

Output:-



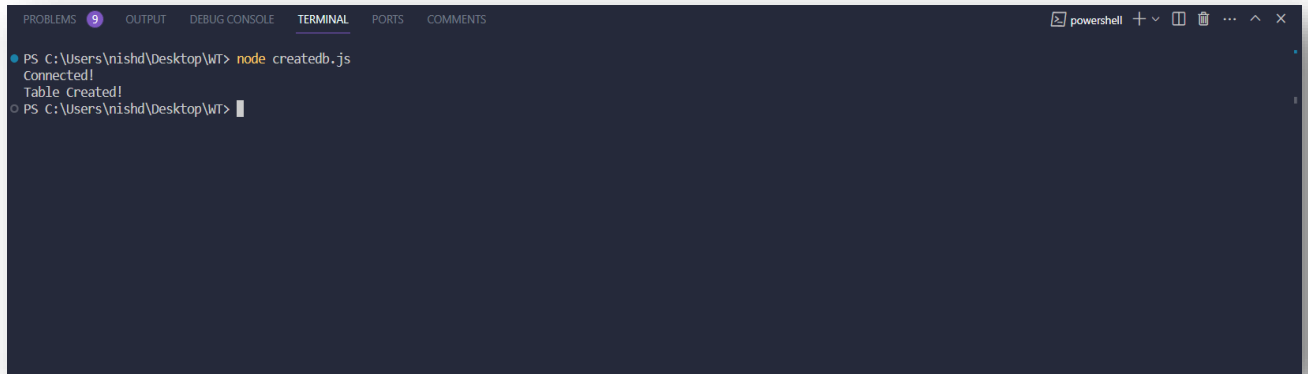
```
PS C:\Users\nishd\Desktop\WT> node createdb.js
Connected!
Database created successfully!
PS C:\Users\nishd\Desktop\WT>
```



Create table:

```
var mysql=require('mysql');
var
    con=mysql.createConnection
    ({host:'localhost',
    user:'root',
    password:'',
    database:'DB01'
});
con.connect(function(err){
    if(err) throw err;
    console.log("Connected!
    ");
    var query="CREATE TABLE DB01(Name VARCHAR(10), ADDRESS
    VARCHAR(255)
    con.query(query,function(err,res
    ult){if(err) throw err;
    console.log("Table created!");
    });
});
```

Output:-



```
PS C:\Users\nishd\Desktop\WT> node createdb.js
Connected!
Table Created!
PS C:\Users\nishd\Desktop\WT>
```



Insert table:

```
var mysql=require('mysql');
var
    con=mysql.createConnection
    ({host:'localhost',
    user:'root',
    password:'',
    database:'DB01'
});
con.connect(function(err)
    { if(err) throw err;
    console.log('Connected!
    ');
    var query="INSERT INTO
    DB01(NAME,ADDRESS)
    VALUES('TCS', 'MUMBAI')";
    con.query(query,function(err,res
        ult){if (err) throw err;
        console.log('1 record inserted!');
    });
});
```

Output:-

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

powershell + - [] ... ^ x

PS C:\Users\nishd\Desktop> node createdb.js
Connected!
1 record inserted!
PS C:\Users\nishd\Desktop>

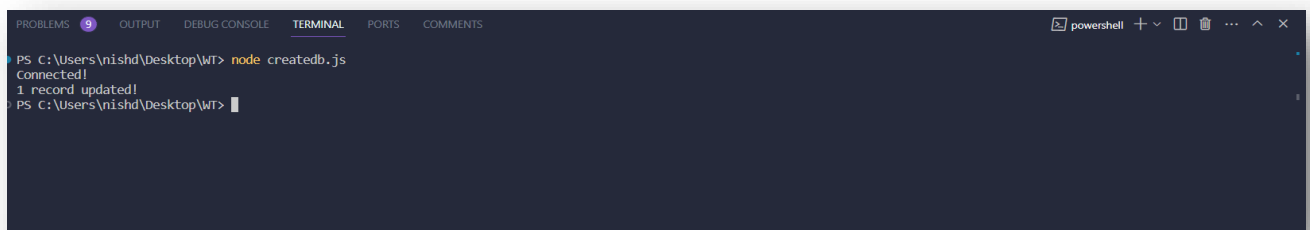
The screenshot shows the phpMyAdmin interface. On the left, the database structure tree is visible, showing a database named 'db01' containing a table named 'customers'. The 'customers' table structure is displayed, showing columns: 'id' (int, primary key), 'name' (varchar, NULL, nullable), and 'address' (varchar, NULL, nullable). The 'New' button is highlighted next to the 'customers' table.

The main panel shows the 'Structure' tab selected. A message indicates: 'Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete'. Below this, a green message states: 'Showing rows 0 - 0 (1 total, Query took 0.0009 seconds.)'. The SQL query entered is: `SELECT * FROM `customers``. The query result is shown as a table with two columns: 'name' and 'address'. The data row shows 'TCS' for 'name' and 'Mumbai' for 'address'.

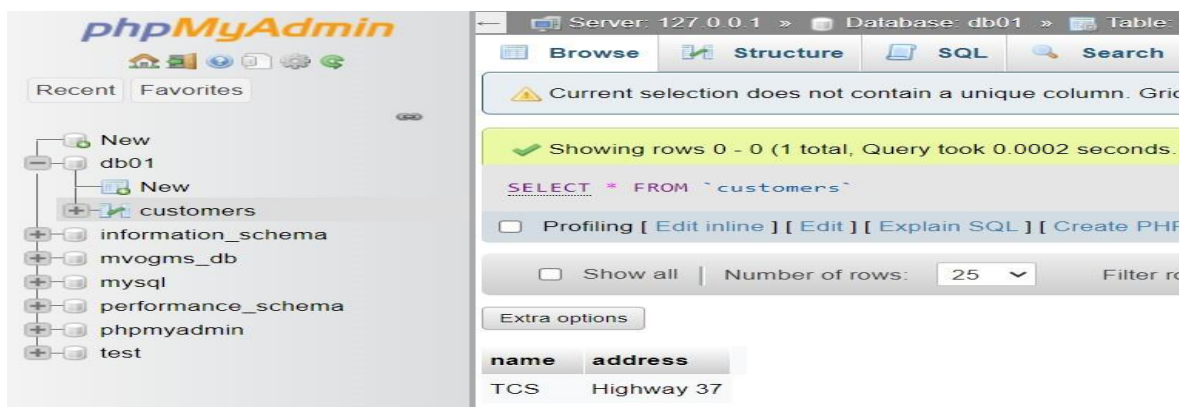
Update table:

```
var mysql=require('mysql');
var
  con=mysql.createConnection
  ({host:'localhost',
   user:'root',
   password:'',
   database:'DB01'
});
con.connect(function(err)
  { if (err) throw err;
    console.log('Connected
    !');
    var query="UPDATE CUSTOMERS SET address='HIGHWAY 37'
    WHERE address=MUMBAI";
    con.query(query,function(err,result){
      if(err) throw err;
      console.log('1 record updated!');
    });
  });
});
```

Output:-



A screenshot of a Windows PowerShell terminal window. The title bar shows 'powershell' with standard window controls. The terminal content shows the following sequence: a command prompt 'PS C:\Users\nishd\Desktop\WT>' followed by the command 'node createdb.js'. The output is 'Connected!' followed by '1 record updated!'. The prompt returns to 'PS C:\Users\nishd\Desktop\WT>'.



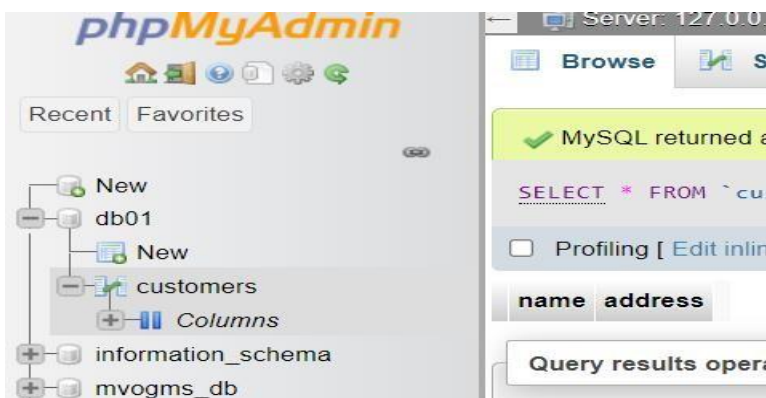
Delete record:

```
var mysql=require('mysql');
var
    con=mysql.createConnection
    ({host:'localhost',
    user:'root',
    password:'',
    database:'DB01'
});
con.connect(function(err)
    { if (err) throw err;
    console.log('Connected
    !');
    var query="DELETE FROM DB01 WHERE
    address=HIGHWAY 37";
    con.query(query,function(err,result){
        if(err) throw err;
        console.log('1 record deleted!');
    });
});
```

Output:-



```
PS C:\Users\nishd\Desktop\WT> node createdb.js
Connected!
1 record deleted!
PS C:\Users\nishd\Desktop\WT>
```



7) Create an Application using filters

Code:

```
<!DOCTYPE HTML>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/an
gularjs/1.8.2/angular.min.js"></script>
</head>
<body style="text-align:center;">
<div ng-app="app">
<div ng-controller="controller">
<form action="javascript:void(0)"> Name:
<input type="text"
ng-model="user.fName" /><br> Email :
<input type="text"
ng-model="user.lName" /><br>
Salary : <input type="text" ng-
model="user.sal"><br> DOB : <input
type="date" ng-model="user.dob">
<br><br>
<button ng-click="getData(user)"> SUBMIT
</button>
</form>
<p><b>Name :</b>
{{userData.fName|uppercase}}<br>
<b>Email :</b>
{{userData.lName|lowercase}}<br>

<b> Salary:</b> {{userData.sal|currency:"₹
```

```

    "}}<br>
    <b>Date of Birth:</b> {{ userData.dob|date :
    "dd.MM.y"}}<br>
    <ul>
    <b>Courses Completed :</b>
    <li ng-repeat= "x in courses | orderBy :
    'name'">
    {{x.name + ", " + x.platform}}
    </li>
    </ul></p>
    </div>
    </div>
    </body>
    <script>
    var myApp = angular.module("app", []);
    myApp.controller("controller", function
    ($scope) {
    $scope.userData = "";
    $scope.getData = function (user) {
    $scope.userData = angular.copy(user);
    $scope.courses = [{ "name": "React Native",
    "platform": "Udemy"
    }, {
    "name": "Data Engineering",
    "platform": "Coursera"
    }, {
    "name": "Bootcamp in Project Management",
    "platform": "Udemy"
    },];
    };
    });
    </script>
    </html>

```

Output:

Before Submit:

The screenshot shows a web browser window with the address bar displaying 'C:/Users/nishd/Desktop/WT/filters.html'. The page content is an empty HTML form with the following fields and labels:

- Name:
- Email:
- Salary:
- DOB:
-
- Name :**
- Email :**
- Salary:**
- Date of Birth:**
- Courses Completed :**

After Submit:

The screenshot shows the same web browser window after the form has been submitted. The fields are now populated with the following data:

- Name: Nishant Deepak Desai
- Email: nish@gmail.com
- Salary: 10000
- DOB: 12/04/2002
-

Below the input fields, the data is displayed in a structured format:

- Name :** NISHANT DEEPAK DESAI
- Email :** nish@gmail.com
- Salary:** ₹ 10,000.00
- Date of Birth:** 04.12.2002

Under the 'Courses Completed :' label, there is a list of courses:

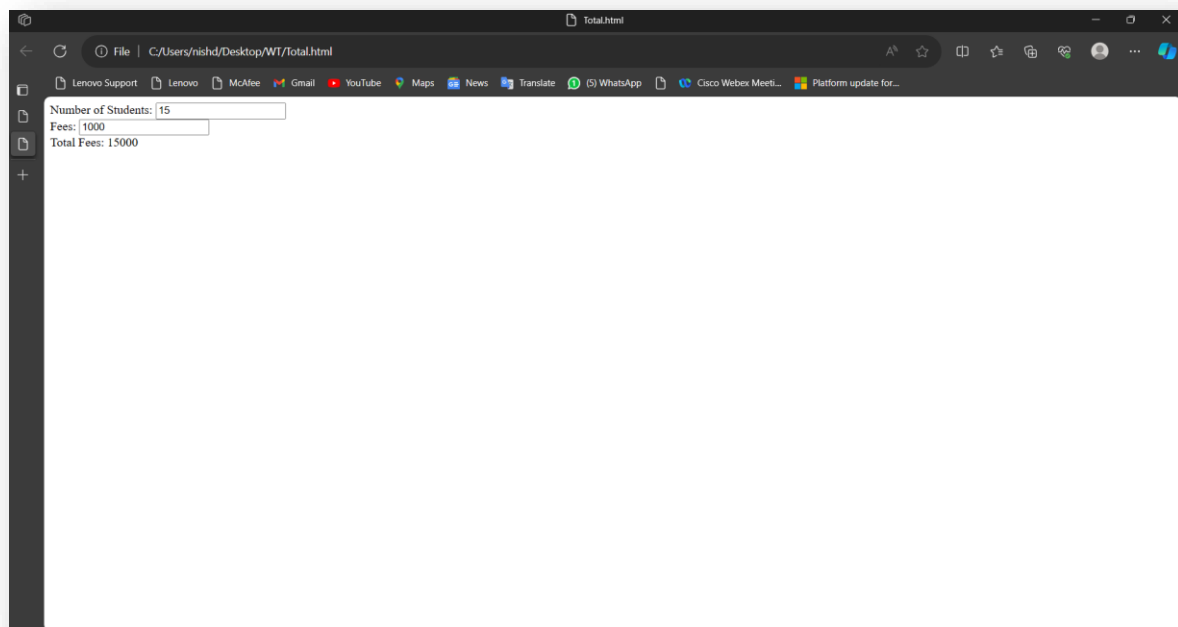
- Bootcamp in Project Management, Udemy
- Data Engineering, Coursera
- React Native, Udemy

8) Create an Application to Demonstrate directives

Code:

```
<html>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <body>
    <div ng-app="" ng-init="students=15;fees=1000">
      Number of Students: <input type="number" ng-model="students" /> <br />
      Fees: <input type="number" ng-model="fees" /> <br />
      Total Fees: {{ students * fees }}
    </div>
  </body>
</html>
```

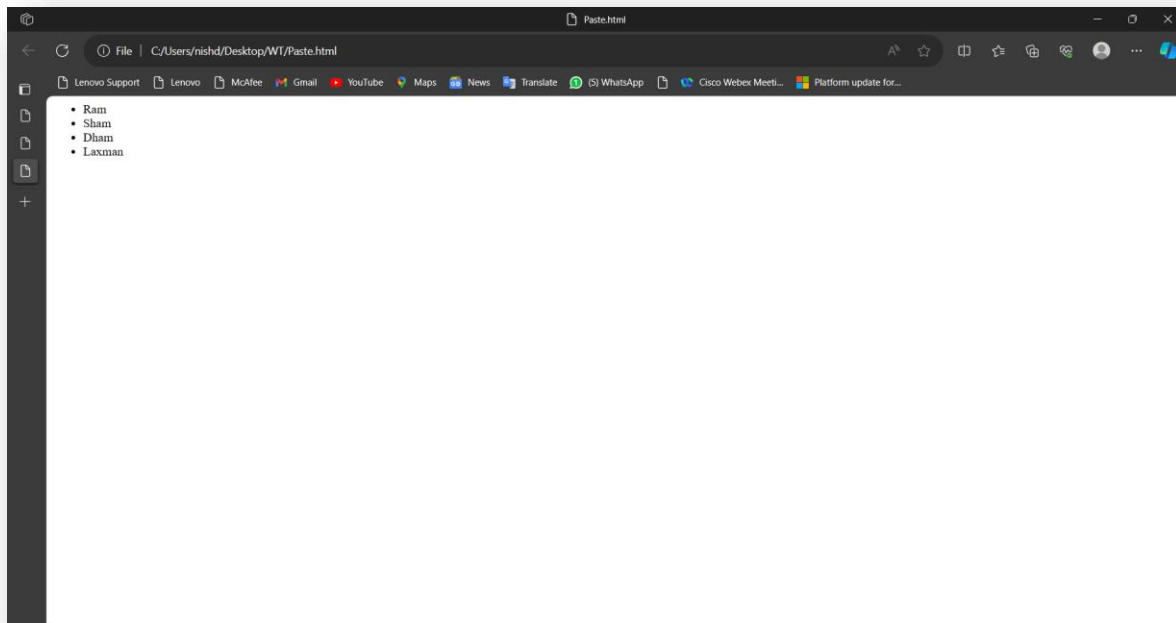
Output:



Code:

```
<html>
  <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
  <body>
    <div ng-app="" ng-init="names=['Ram','Sham','Dham','Laxman']">
      <ul>
        <li ng-repeat="x in names">
          {{ x }}
        </li>
      </ul>
    </div>
  </body>
</html>
```

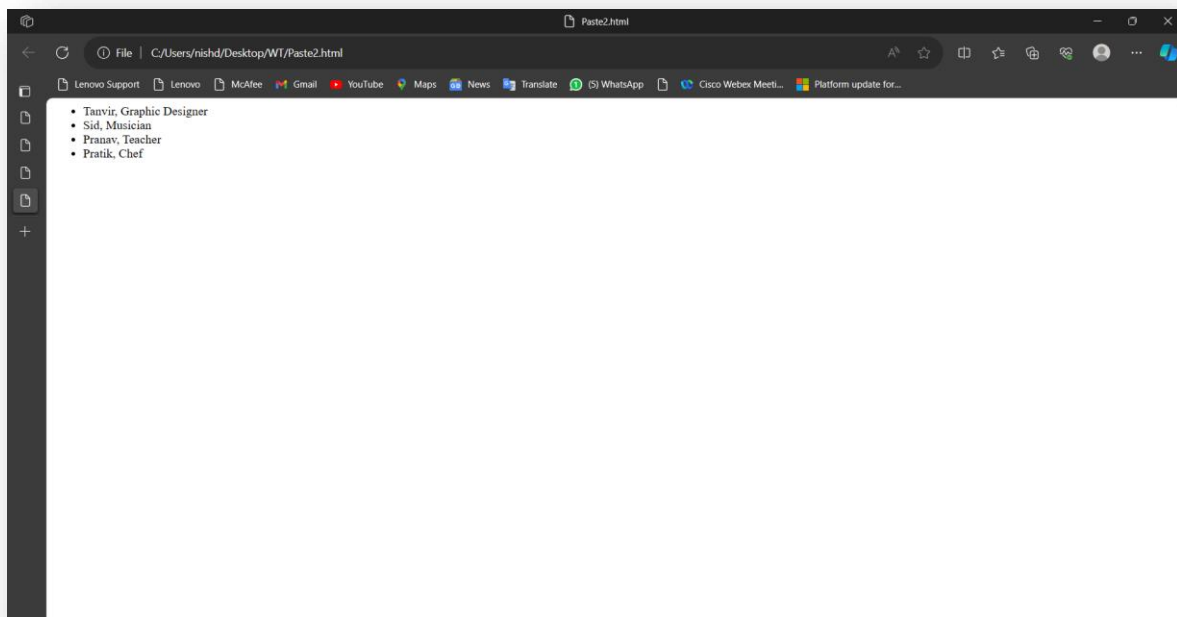
Output:



Code:

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
  <div ng-app="" ng-init="names=[
    {name:'Tanvir',proffesion:'Graphic Designer'},
    {name:'Sid',proffesion:'Musician'},
    {name:'Pranav',proffesion:'Teacher'},
    {name:'Pratik',proffesion:'Chef'}]">
    <ul>
      <li ng-repeat="x in names">
        {{ x.name + ', ' + x. proffesion: }}
      </li>
    </ul>
  </div>
</body>
</html>
```

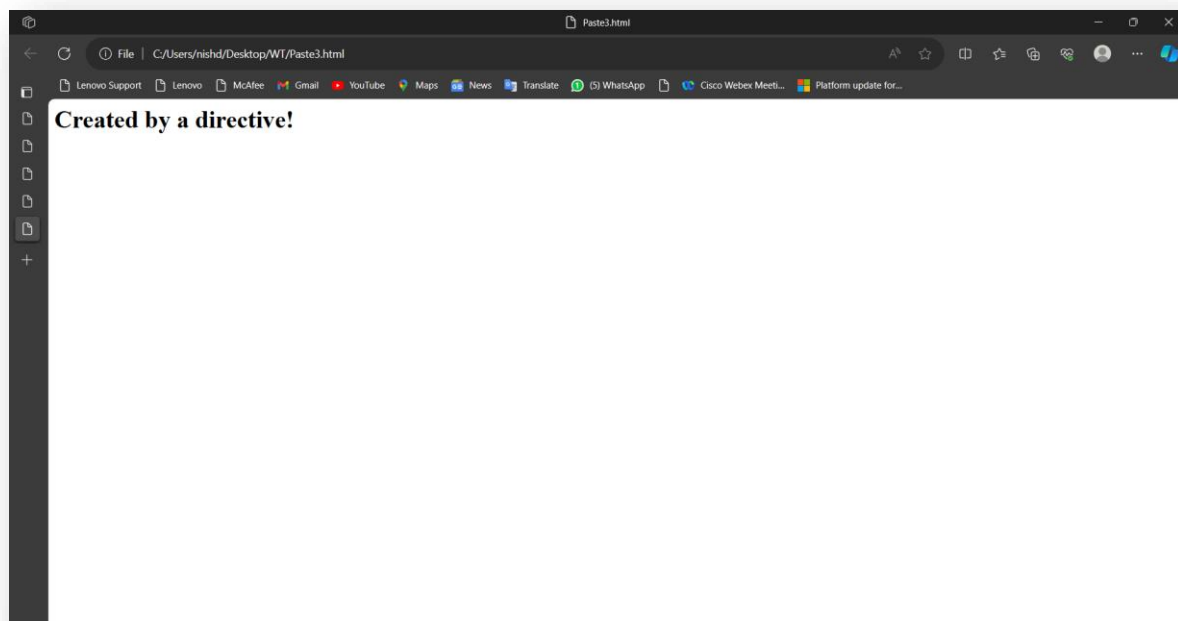
Output:



Code:

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
  <body ng-app="myApp">
    <w3-test-directive></w3-test-directive>
    <script>
      var app = angular.module("myApp", []);
      app.directive("w3TestDirective",
      function() {return {
      template : "<h1>Created by a directive!</h1>"
      };
      });
    </script>
  </body>
</body>
</html>
```

Output:

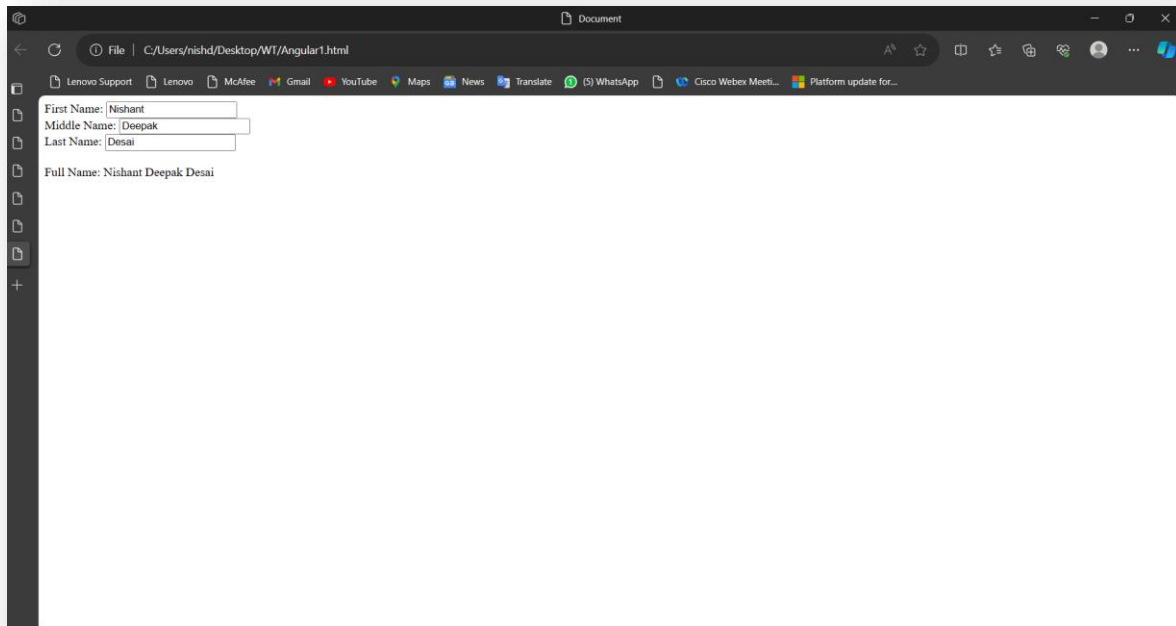


9) Demonstrate controllers in Angular.js through an application

Code:

```
<!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>Document</title>
  </head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <body>
    <div ng-app="myApp" ng-controller="myCtrl">
      First Name: <input type="text" id="fname" ng-model="firstname" /><br />
      Middle Name: <input type="text" ng-model="middlename" /><br />
      Last Name: <input type="text" ng-model="lastname" /><br />
      <br />
      Full Name: {{firstname+ " "+middlename+ " "+lastname}}
    </div>
    <script>
      var app = angular.module("myApp", []);
      app.controller("myCtrl", function ($scope) {
        $scope.firstname = "Nishant";
        $scope.middlename = "Deepak";
        $scope.lastname = "Desai";
      });
    </script>
  </body>
</html>
```

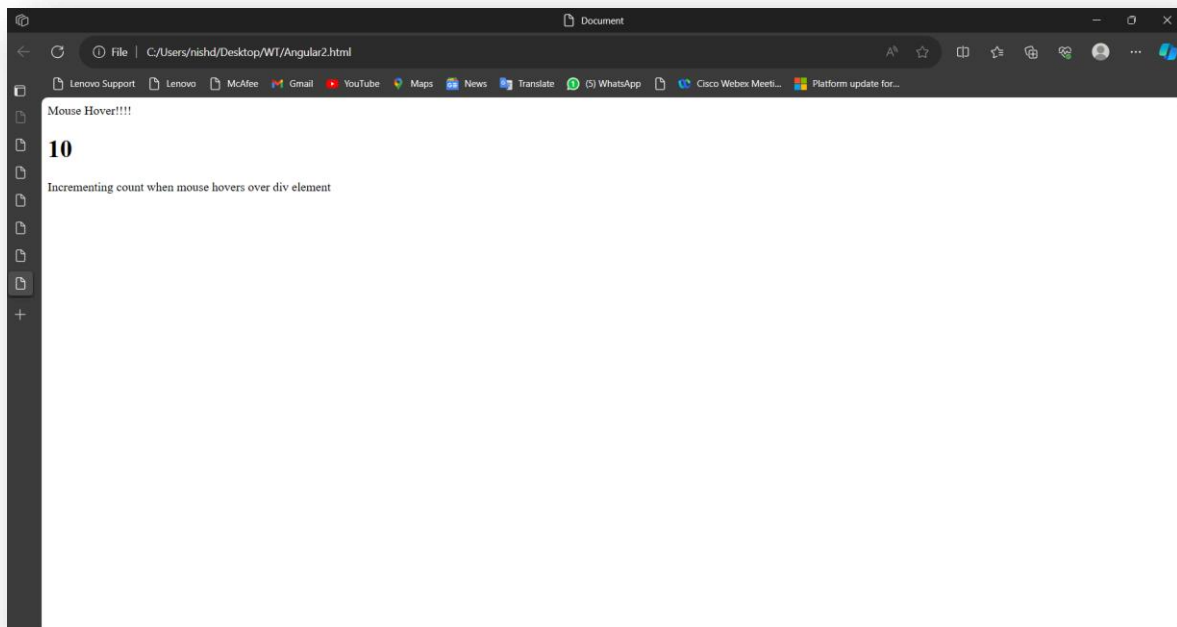
Output:



Code:

```
<!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>Document</title>
</head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
<body ng-app="">
  <div ng-mouseenter="count=count+1" ng-init="count=0">Mouse Enter!!!</div>
  <h1>{{count}}</h1>
  <p>Incrementing count when mouse hovers over div element</p>
</body>
</html>
```

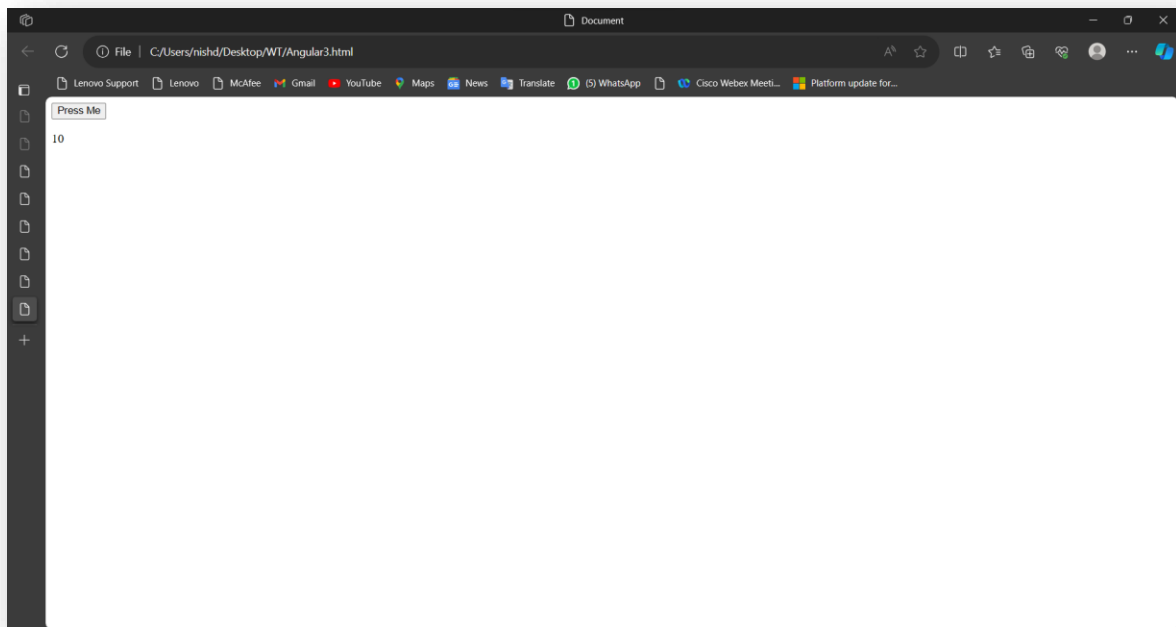
Output:



Code:

```
<!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>Document</title>
</head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<button ng-click="count=count+1">Click Me</button>
<p>{{count}}</p>
</div>
<script>
var app=angular.module('myApp',[]); app.controller('myCtrl',function($scope){
$scope.count=0
});
</script>
</body>
```

Output:



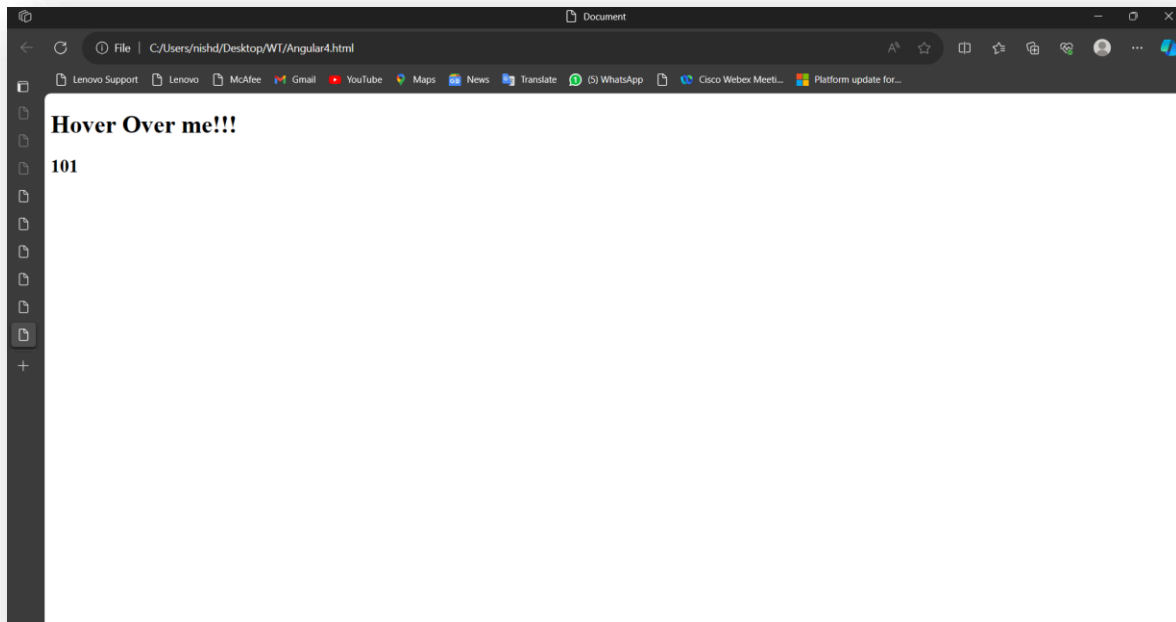
Code:

```
<!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>Document</title>
</head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
<body>
  <div ng-app="myApp" ng-controller="myCtrl">
    <h1 ng-mousemove="count=count+1">Hover Over me!!!</h1>
    <h2>{{count}}</h2>
  </div>

  <script>
    angular.module('myApp', []).controller('myCtrl',function($
scope){
  $scope.count=0;
  })
```

```
</script>  
</body>  
</html>
```

Output:



10) Demonstrate features of Angular.js froms with a program

Code:

```
<html>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
  <style>
    input.ng-invalid {
      background-color: lightcoral;
    }
    input.ng-valid {
      background-color: white;
    }
  </style>
  <body ng-app="myApp">
    <center>
      <form name="userdashboard">
        <fieldset style="width: 40%">
          <legend><h2>User DashBoard</h2></legend>
          <div ng-controller="formCtrl">
            <h3>
              <b>
                >Enter Full Name:
```

[illegible]

```
<input
  type="text"
  name="name"0
  style="padding: 5px"
  ng-model="fullname"
/>
</b>
</h3>

<h3>
  <b
    >Apartment Number:
```

[illegible]


```
<input
  type="text"
  name="numfamily"
  style="padding: 5px"
  ng-model="familynumbers"
/>
</b>
</h3>
<p>
  The input's valid state
  is:{{userdashboard.apartmentnumber.$valid}}
</p>
</div>
<h3>
  <b>
    >Type of Vehicles:
```

[illegible]

```
<!-- <input type="number" name="numvehicle" style="padding: 5px" /> -->
</b>
<select ng-model="myVar">
  <option value=""></option>
  <option value="Bike">Bike</option>
  <option value="Car">Car</option>
  <option value="Truck">Truck</option>
</select>
</h3>
<h3>
<b
  >Type: &nbsp;
  <input
    type="radio"
    name="numfamily"
    style="padding: 5px"
    ng-model="type"
  />Owner
  <input

    type="radio"
```

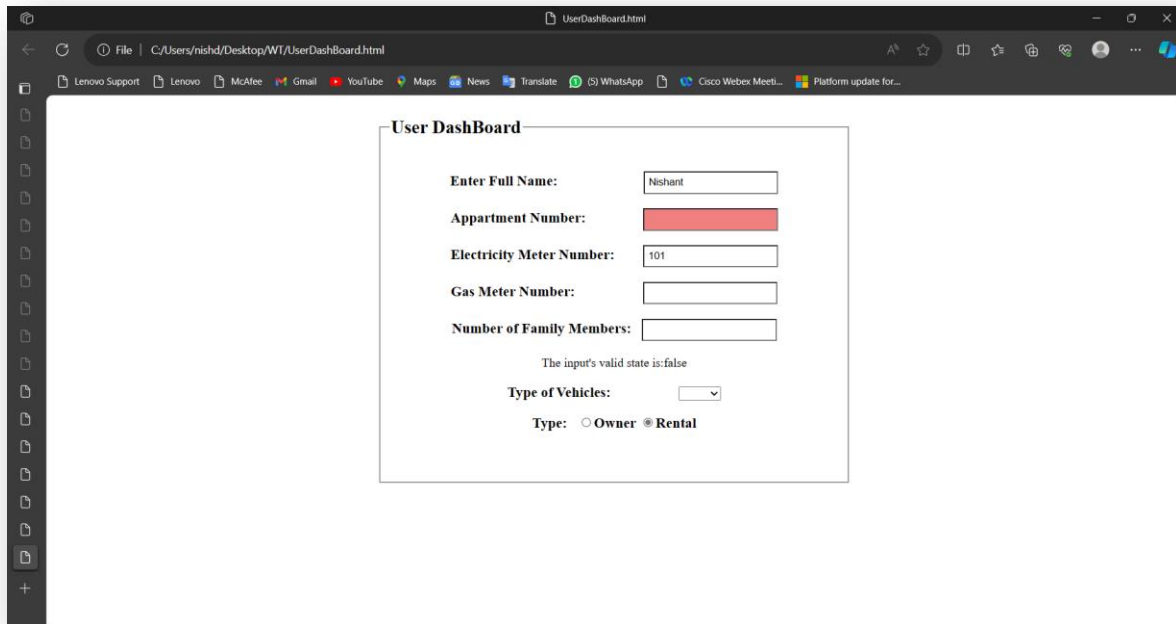
```

        name="numfamily"
        style="padding: 5px"
        ng-model="type"
    />Rental
</b>
</h3>
<div ng-switch="myVar">
    <div ng-switch-when="Bike">
        <h1>Bike</h1>
        <p><b>1 Tier Parking</b></p>
    </div>
    <div ng-switch-when="Car">
        <h1>Car</h1>
        <p><b>2 Tier Parking</b></p>
    </div>
    <div ng-switch-when="Truck">
        <h1>Truck</h1>
        <p><b>4 Tier Parking</b></p>
    </div>
</div>
<br /><br />
</fieldset>
</form>
</center>
<script>
    var app = angular.module("myApp", []);
    app.controller("formCtrl", function ($scope) {
        $scope.fullname = "Nishant";
        $scope.meternumber = "101";
    });
</script>
</body>
</html>

```

Output:

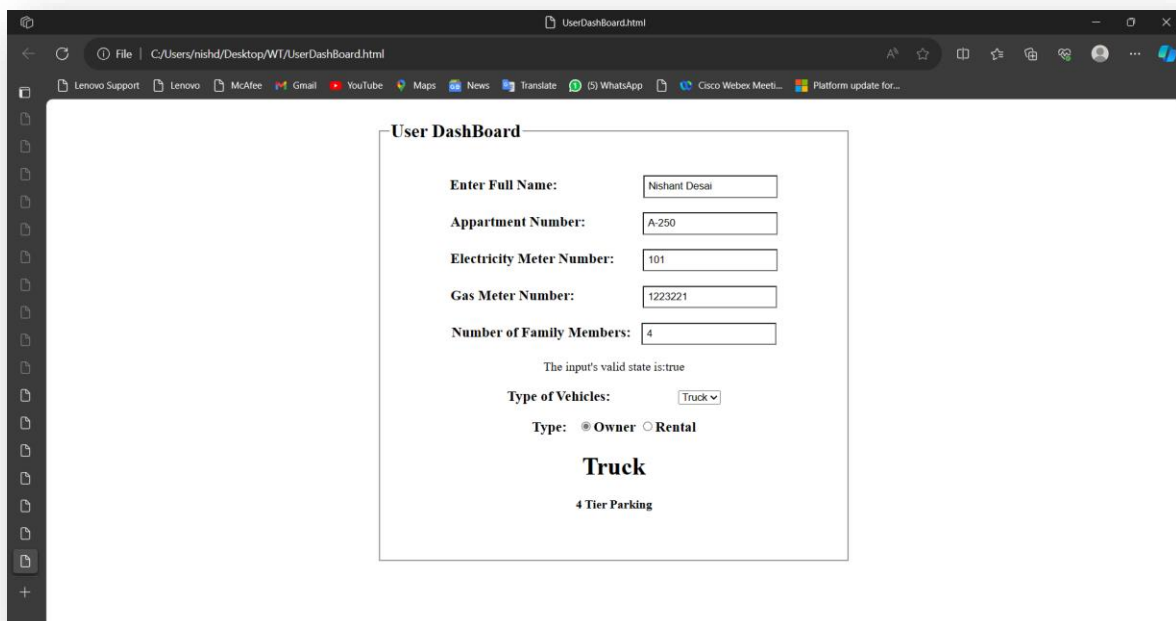
Before Input:



The screenshot shows a web browser window titled "UserDashboard.html". The address bar shows the file path "C:/Users/nishd/Desktop/WT/UserDashboard.html". The browser's tab bar includes "Lenovo Support", "Lenovo", "McAfee", "Gmail", "YouTube", "Maps", "News", "Translate", "(S) WhatsApp", "Cisco Webex Meet...", and "Platform update for...". The main content area displays a form titled "User Dashboard". The form contains the following fields and labels:

- Enter Full Name:
- Appartment Number:
- Electricity Meter Number:
- Gas Meter Number:
- Number of Family Members:
- The input's valid state is: false
- Type of Vehicles:
- Type: ☐ Owner ☒ Rental

After Input:



The screenshot shows the same web browser window as before, but with the form fields filled out. The form title is "User Dashboard". The fields and their values are:

- Enter Full Name:
- Appartment Number:
- Electricity Meter Number:
- Gas Meter Number:
- Number of Family Members:
- The input's valid state is: true
- Type of Vehicles:
- Type: ☒ Owner ☐ Rental

Below the form, the text "Truck" is displayed in a large, bold font, and "4 Tier Parking" is displayed in a smaller font below it.

11) Create a SPA (Single Page Application)

Code:

```
<!DOCTYPE html>
<!--ng-app directive tells AngularJS that myApp is the root element of the application -->
<html ng-app="myApp">
  <head>
    <!--import the angularjs libraries-->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-
route.min.js"></script>

    <style>
      body {
        text-align: center;
        font-family: Arial, Helvetica, sans-serif;
        background-color: white;
      }

      h1 {
        color: black;
      }
    </style>
  </head>

  <body>
    <h1>ATHLETICS FEDERATION OF INDIA</h1>
    <!--hg-template indicates the pages that get loaded as per requirement-->
    <script type="text/ng-template" id="first.html">
      <h1>Cricket</h1>
      <h2 style="color:black"> Cricket Where Skills Create Music!
    </h2>
      <h3>{{message}}</h3>
    </script>
    <script type="text/ng-template" id="second.html">

      <h1>Football</h1>
      <h2 style="color:black">
        Game of Warriors<br> Apply now!
      </h2>
```

```
<h3>{{message}}</h3>
</script>
```

```
<script type="text/ng-template" id="third.html">
  <h1>Hockey</h1>
  <h2 style="color:black">
    Game of Legends<br> Apply now!
  </h2>
  <h3>{{message}}</h3>
</script>
```

```
<!-- Hyperlinks to load different pages dynamically -->
<a href="#/">Cricket</a>
<a href="#/second">Football</a>
<a href="#/third">Hockey</a>
```

```
<!--ng-view includes the rendered template of the current route into the main page-->
<div ng-view></div>
```

```
<script>
var app = angular.module("myApp", []);
var app = angular.module("myApp", ["ngRoute"]);
app.config(function ($routeProvider) {
  $routeProvider
    .when("/", {
      templateUrl: "first.html",
      controller: "FirstController",
    })

    .when("/second", {
      templateUrl: "second.html",
      controller: "SecondController",
    })
```

```
    .when("/third", {
      templateUrl: "third.html",
      controller: "ThirdController",
    })
```

```
.otherwise({ redirectTo: "/" });  
});  
  
app.controller("FirstController", function ($scope) {  
    $scope.message = "Message From Cricket Federation";  
});  
app.controller("SecondController", function ($scope) {  
    $scope.message = "Message From Football Department";  
});  
app.controller("ThirdController", function ($scope) {  
    $scope.message = "Message From Hockey Department";  
});  
</script>  
</body>  
</html>
```

Output:

