Web Technologies Journal

Tejas Khobrekar (C-23057)

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. **Tejas A. Khobrekar** of M.C.A. Semester I with Roll No. **C23057** has completed _____ practicals of Web Technologies under my supervision in this college during the year 2023-2024.

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

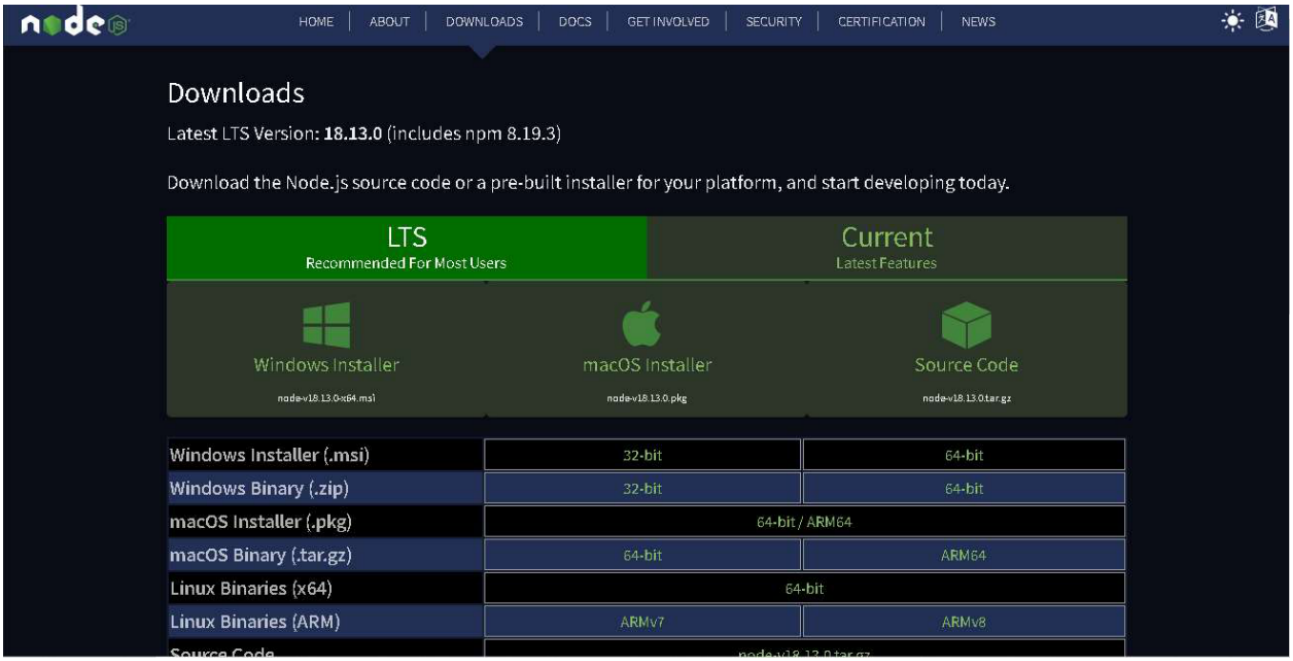
Sr.No	Topic Name	Date	CO	Sign
1	Create an application to demonstrate Node.js Modules		CO1	
2	Create an application to demonstrate various Node.js Events		CO1	
3	Create an application to demonstrate Node.js Functions		CO1	
4	Using File Handling demonstrate all basic file operations (Create, write, read, 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	

- **Steps to Install Node.js Application:-**

Before proceeding, let's first install Node.js on out system ~

Step 1: Obtain the Installer
Visit the official Node.js website and download the Windows Installer. Ensure that you acquire the latest Node.js version, which comes bundled with the NPM package manager. Opt for the

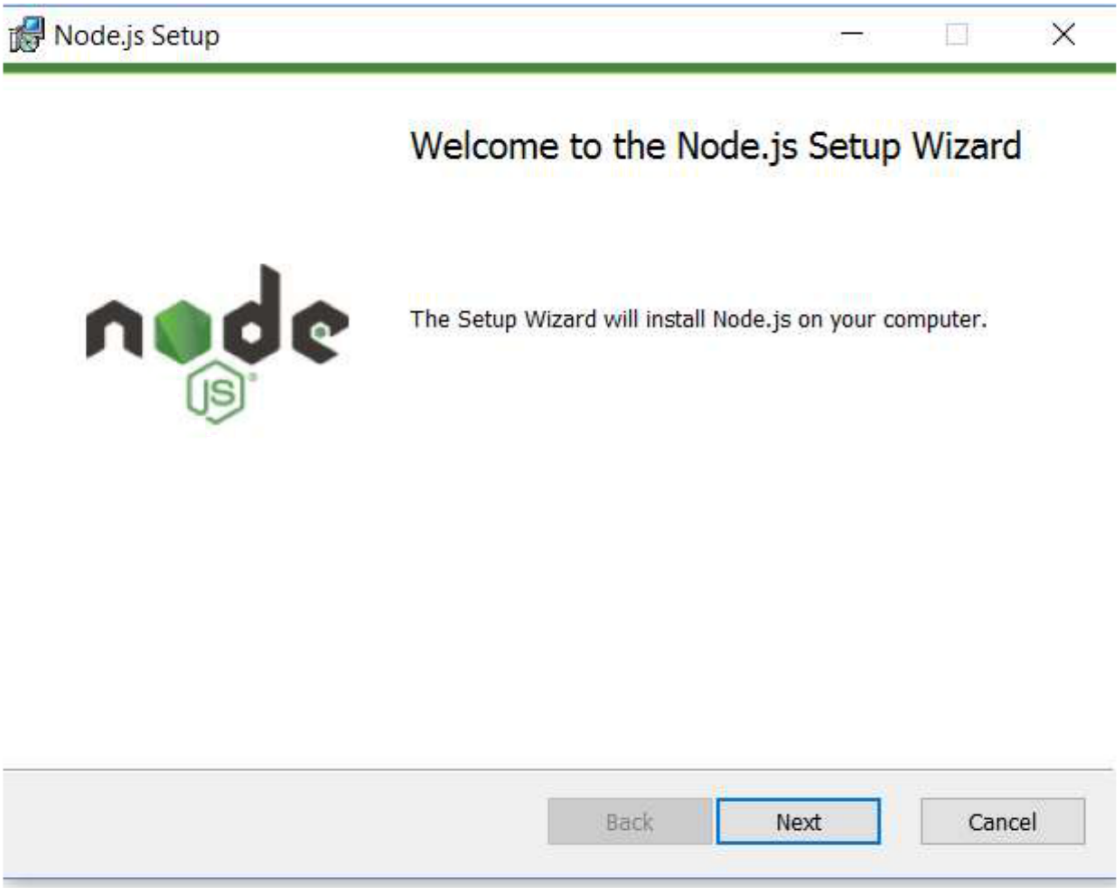
64-bit version of the Node.js installer.



Upon completing the download, execute the installation by double-clicking the downloaded installer package. A .msi file will be saved to your browser, allowing you to choose the preferred installation location.

Step 2: Node.js Installation

- Double-click on the .msi binary files in the chosen path to commence the installation.
- Grant the necessary permissions to run the application.
- A welcome message will appear; click the "Next" button.
- The installation process will initiate.
- Select the desired installation path for Node.js.



Step 3: Verify Node.js Installation
Once Node.js is installed on your system, confirm its installation by opening the command prompt and typing `node --version`. A successful installation will display the installed Node.js version.

Post-installation, you can select an Integrated Development Environment (IDE) for Node.js development. Recommended IDEs include Visual Studio, Sublime Text, Eclipse, etc.

Practical 1

Aim - Create an application to demonstrate Node.js Modules.

Theory - Within Node.js, Modules serve as encapsulated code units that interact with external applications based on their specific functionalities. Modules can take the form of a single file or a compilation of multiple files and folders.

To incorporate a module, employ the `require()` function along with the module's name. For instance:

var http = require('http');

Following this, our application gains access to the HTTP module, enabling the creation of a server.

Commonly utilized modules include:
1. http
2. fs
3. express
4. event
5. net

Code -

firstModule.js

```
exports.currentDate = function () {
 return Date();
}
```

usingFirstModule.js

```
var myModule = require('./firstModule');

console.log(myModule.currentDate());
```

In the provided illustration, two modules, mymodule and usemodule, have been established. Within mymodule, a function has been defined to retrieve the current date using Date(). Subsequently, in usemodule, the require() keyword is employed to import the module. Upon execution of the usemodule.js script, the output is as follows:

Output -

```
"C:\Program Files\nodejs\node.exe" ".\Practical 1.js"
Sun Nov 26 2023 11:55:32 GMT+0530 (India Standard Time)
```

Practical 2

Aim - Create an application to demonstrate various Node.js Events

Theory - An event emitter is an object designed to emit events, while an event listener is a component of the code that attaches to the event emitter and awaits specific types of events.

Code -

EventEmitter.js

```
const events = require("events");
const eventEmitter = new events.EventEmitter();

function listener1() {
  console.log("Event received by Listener 1");
}

function listener2() {
  console.log("Event received by Listener 2");
}

eventEmitter.addListener("write", listener1);
eventEmitter.on("write", listener2);

eventEmitter.emit("write");
console.log(eventEmitter.listenerCount("write"));

eventEmitter.removeListener("write", listener1);
console.log("Listener 1 is removed");

eventEmitter.emit("write");

console.log(eventEmitter.listenerCount("write"));
console.log("Program Ended");
```

Output -

```
"C:\Program Files\nodejs\node.exe" ".\practical 2.js"
Event received by Listener 1
Event received by Listener 2
2
Listener 1 is removed
Event received by Listener 2
1
Program Ended
```

Practical 3

Aim - Create an application to demonstrate NodeJs functions.

Theory - Functions are composed of a return statement; if absent, the function defaults to returning undefined. Once defined, a function can be invoked, initiating its execution.

Structure of a function -

```
function functionName(parameters) {
  // body of the function
  // return statement
}
```

Code -

CallbackFunction.js

```
function displayResult(parameter) {
    console.log(parameter);
}

function calculate(x, y, myCallback) {
    let sum = x + y;
    myCallback(sum);
}

calculate(8, 10, displayResult);
```

Output -

```
"C:\Program Files\nodejs\node.exe" ".\practical 3.js"
18
```

Theory - Arrow functions offer a concise syntax for writing functions in JavaScript. Unlike other function types, the value of `this` within arrow functions is not dependent on how they are invoked or defined; it relies solely on their enclosing context.

Code -

```
const message = function () {
  console.log("Hi, I am Tejas Khobrekar 🗿");
};

setTimeout(message, 3000);

setTimeout(() => {
  console.log("Calling arrow function.");
}, 5000);
```

Output -

```
C:\Program Files\nodejs\node.exe .\Documents\vb\3.js
Hi, I am Tejas Khobrekar 🗿
Calling arrow function.
```

Practical 4

Aim - Create an HTTP server and perform operations.

Theory - Accessing web pages in any web application necessitates a web serve to manage HTTP requests. Different web servers are employed for various web application technologies; for instance, IIS for ASP.NET applications and Apache for PHP or Java applications. In the context of Node.js, it provides the capability to create custom web servers, enabling the asynchronous handling of HTTP requests.

Node.js simplifies the process of establishing a basic web server that can efficiently process incoming requests asynchronously. The following example showcases a straightforward Node.js web server encapsulated in the Prac5A.js file.

Code -

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

var server = http.createServer(function (req, res) {

res.write("Hello Server");

res.end();

});

server.listen (5000);

console.log('Node.js web server at port 5000 is running!');
```

Output -

```
"C:\Program Files\nodejs\node.exe" ".\var http require('http');.js"
Node.js web server at port 5000 is running!
```

Handling HTTP Requests:

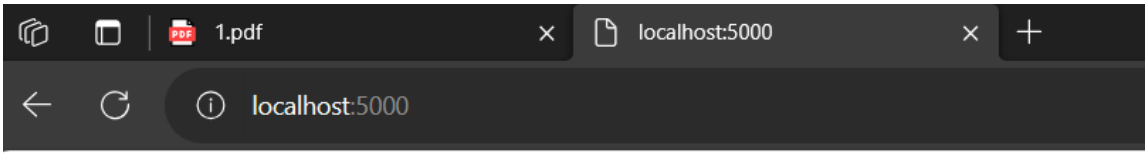
The `http.createServer()` method incorporates `request` and `response` parameters, supplied by Node.js. The `request` object provides information about the current HTTP request, such as the URL, request header, and data. The `response` object is utilized to send a response for the current HTTP request.

Code -

```
var http = require('http');
var server = http.createServer(function (req, res) {
  if (req.url == '/') {
    res.writeHead(200, { 'Content-type': 'text/html' });
    res.write('<html><body><p>This is the home page.</p></body></html>');
    res.end();
  } else if (req.url == "/student") {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is the student page.</p></body></html>');
    res.end();
  } else if (req.url == "/admin") {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is the admin page.</p></body></html>');
    res.end();
  } else {
    res.end('Invalid Request!');
  }
});
server.listen(5000);
console.log('Node.js web server at port 5000 is running...');
```

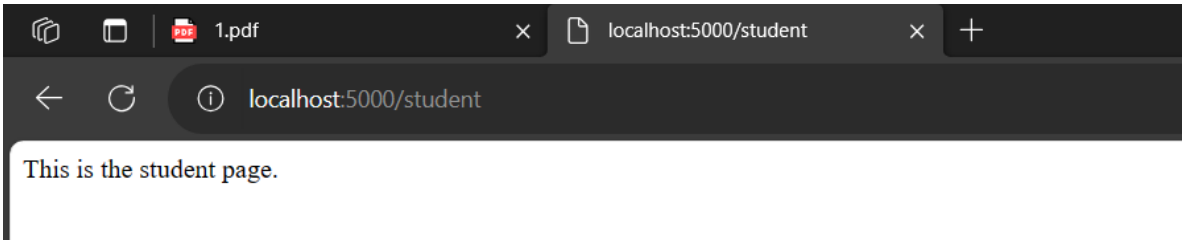
Output -

When accessing "localhost:5000" in the browser, the following output will be displayed:



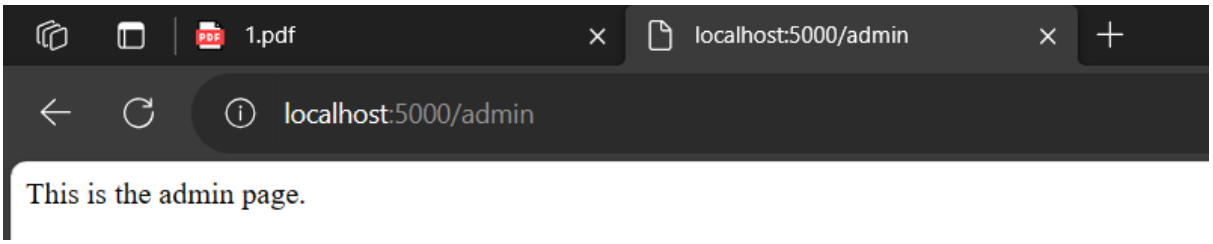
This is the home page.

Upon changing the URL to "localhost:5000/student," the output will be as follows:



This is the student page.

Upon changing the URL to "localhost:5000/admin," the output will be as follows:



This is the admin page.

Practical 5

Aim - Using File Handling to demonstrate all basic file operations (Create, write, read, delete).

Theory - In Node.js, the fs module facilitates access to the physical file system, handling both asynchronous and synchronous file I/O operations. The module supports various file operations, including reading, writing, and deleting files.

1. Reading File:

To asynchronously read a file, the `fs.readFile()` method is employed. The structure is as follows:

```
fs.readFile(fileName[, options], callback)
```

- `fileName`: Full path and name of the file as a string.
- `options`: An object or string containing encoding and flag parameters. Default encoding is UTF8, and the default flag is "r".
- `callback`: A function with parameters `err` and `fd`, called upon completion of the readFile operation.

Code -

```
var fs = require("fs");
fs.readFile("Tejas.txt", function (err, data) {
  if (err) {
    return console.error(err);
  }
  console.log("Asynchronous read: " + data.toString());
});
```

Output -

```
C:\Program Files\nodejs\node.exe .\Practical4.js
Asynchronous read: Hey!!
My name is Tejas Khobrekar,
I love coffee.
```

Use `fs.readFileSync()` method to read files synchronously.

Code -

```
var fs = require("fs");
var data = fs.readFileSync("Tejas.txt");
console.log("Synchronous read: " + data.toString());
console.log("Program Ended");
```

Output -

```
"C:\Program Files\nodejs\node.exe" ".\Practical 4.js"
Synchronous read: Hey!!
My name is Tejas Khobrekar,
I love coffee.
Program Ended
```

2. Writing File:

For writing a file, the `fs.writeFile()` method is employed. The structure is:

```
fs.writeFile(fileName, data[, options], callback)
```

- `fileName`: Full path and name of the file as a string.
- `data`: The content to be written to the file.
- `options`: An object or string containing encoding and flag parameters. Default encoding is UTF8, and the default flag is "r".
- `callback`: A function with parameters `err` and `fd`, called upon completion of the writeFile operation.

Code -

```
var fs = require("fs");
fs.writeFile("Tejas.txt", "Hii Tejas. Hope you're havin' a good day!", function (err) {
  if (err) {
    return console.log(err);
  } else {
    console.log("Write operation complete");
  }
});
```

Output -

```
C:\Program Files\nodejs\node.exe .\Practical_4.js
Write operation complete
```

3. Delete File:

To delete an existing file, the `fs.unlink()` method is used. The structure is:

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

Code -

```
var fs = require("fs");
fs.unlink("Tejas.txt", function () {
  console.log("Deletion Complete.");
});
```

Output -

```
C:\Program Files\nodejs\node.exe .\Practical__4.js
Deletion Complete.
```

Practical 6

Aim - Create an application to establish a connection with the MySQL database and perform basic database operations on it.

Theory - Node.js is compatible with various databases, whether relational or NoSQL. To interact with a database in Node.js, you must install the corresponding drivers for the desired database. The following example demonstrates the creation of a MySQL database and the manipulation of its tables using Node.js.

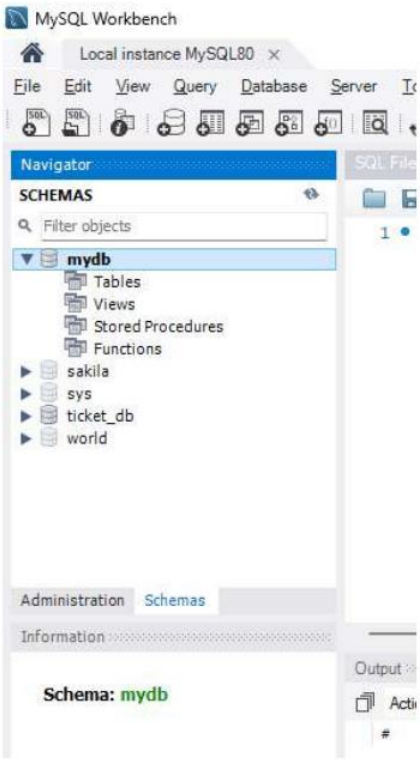
To download and install “mysql” module, open the command terminal and enter the following:

```
PS C:\Users\nehap\Documents\Tejas\vb\nodejs\sem1\practical6> npm install mysql

added 11 packages, and audited 12 packages in 2s

found 0 vulnerabilities
PS C:\Users\nehap\Documents\Tejas\vb\nodejs\sem1\practical6>
```

Code -

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

```
C:\Users\nehap>D:\Tejas\MCA-Course-Archives-main\Sem I\Journal\Web Technologies\Code\Practical 6>node createDatabase.js
Connected!
Database created
```



To confirm the database creation, open the mysql workbench. The screenshot indicates the successful creation of the "mydb" database.

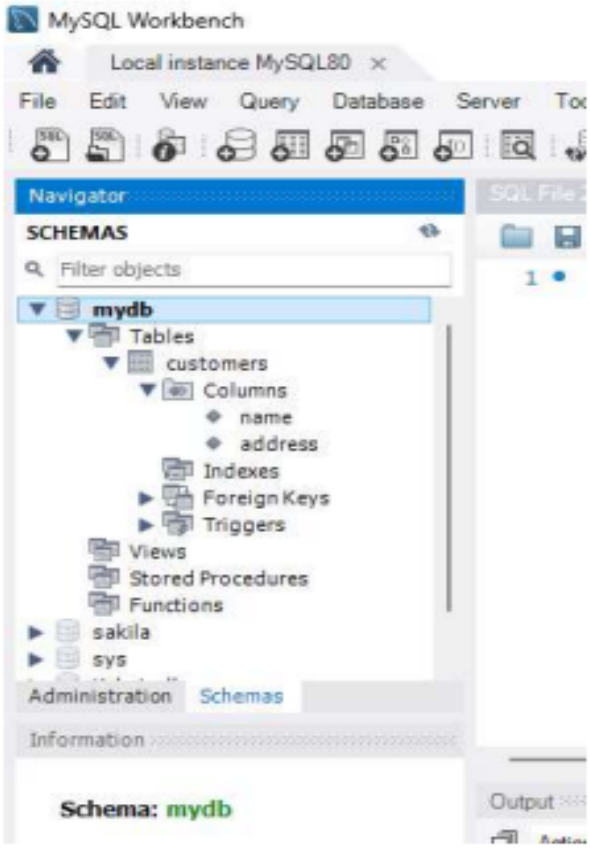
Now, let's proceed to create a table within the "mydb" database:

Code -

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "admin",
  database: "mydb"
});

con.connect(function (err) {
  if (err) throw err;
  console.log("Connected");
  var sql = "CREATE TABLE customers(name VARCHAR(255),
address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("TABLE CREATED");
  });
});
```

Output -

```
C:\Users\nehap>D:\Tejas\MCA-Course-Archives-main\Sem I\Journal\Web Technologies\Code\Practical 6>node createTable.js
Connected
TABLE CREATED
```



The screenshot illustrates the successful creation of the "customers" table within the "mydb" database.

Alter Table:

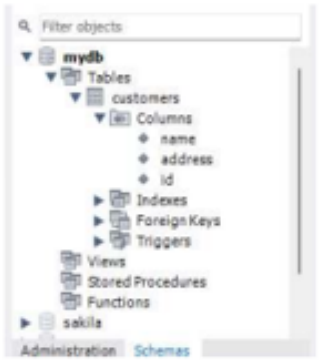
To alter an existing table, execute the following code:

Code -

```
var mysql = require('mysql');
var con = mysql.createConnection({
```

```
  host: "localhost",
  user: "root",
  password: "admin",
  database: "mydb"
});

con.connect(function (err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "ALTER TABLE customers ADD COLUMN id INT
AUTO_INCREMENT PRIMARY KEY";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table altered");
  });
});
```

Output -



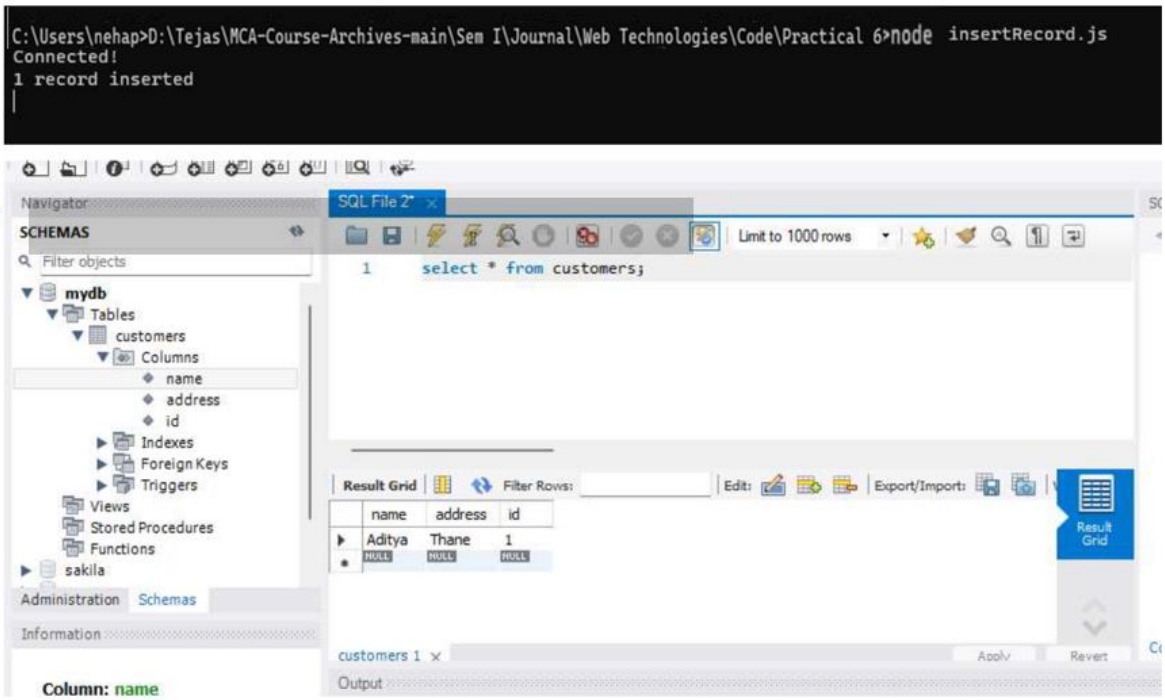
The screenshot reveals the successful allocation of a primary key to the "address" column.

Insert Record:
Insert values into the "customers" table using the following code:

Code -

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "admin",
  database: "mydb"
});

con.connect(function (err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO customers (name, address) VALUES
('Aditya', 'Thane')";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("1 record inserted");
  });
});
```

Output -



The output confirms the successful insertion of one record into the "customers" table.

Read Data:

To retrieve data from the previously created table, execute the following code:

Code -

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "admin",
```

```
  database: "mydb"
});

con.connect(function (err) {
  if (err) throw err;
  con.query("SELECT * FROM customers", function (err, result,
fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

Output -

```
C:\Users\nehap>D:\Tejas\MCA-Course-Archives-main\Sem I\Journal\Web Technologies\Code\Practical  6>node readData.js
[ RowDataPacket { name: 'Aditya', address: 'Thane', id: 1 } ]
```

The output displays the data present in the "customers" table.

Delete Records: Perform a delete operation on the existing record using the following code:

Code -

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "admin",
  database: "mydb"
```

```
});

con.connect(function (err) {
  if (err) throw err;
  var sql = "DELETE FROM customers WHERE address = 'Dahisar
08'";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Number of records deleted: " +
result.affectedRows);
  });
});
```

Output -

```
C:\Users\nehap>D:\Tejas\MCA-Course-Archives-main\Sem I\Journal\Web Technologies\Code\Practical  6>node deleteRecords.js
Number of records deleted: 1
```

The output indicates the number of records deleted, based on the specified condition.

Update Record:

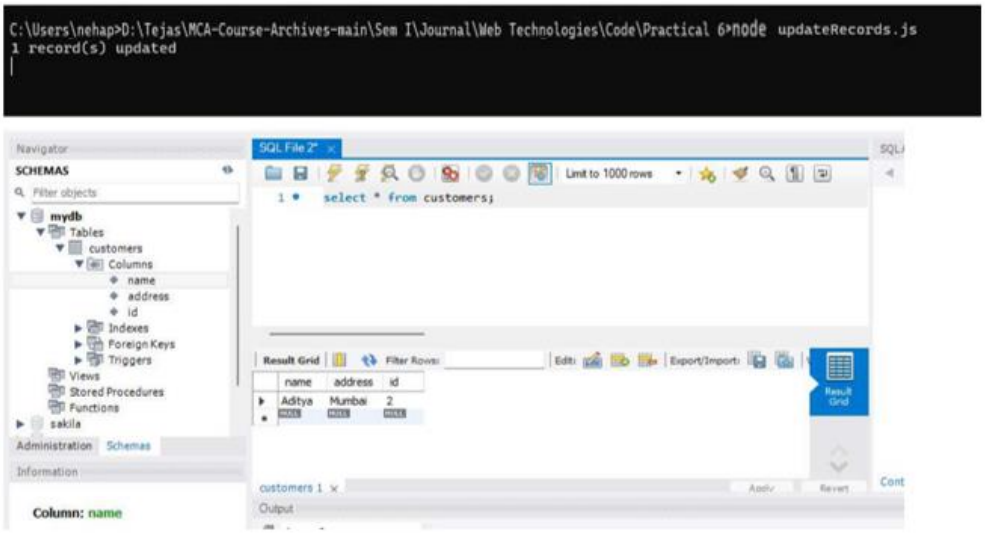
Update a record in the table using the following code:

Code -

```
var mysql = require('mysql');
var con = mysql.createConnection({
```

```
  host: "localhost",
  user: "root",
  password: "admin",
  database: "mydb"
});

con.connect(function (err) {
  if (err) throw err;
  var sql = "UPDATE customers SET address = 'Mumbai' WHERE
address = 'Thane'";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result.affectedRows + " record(s) updated");
  });
});
```

Output -



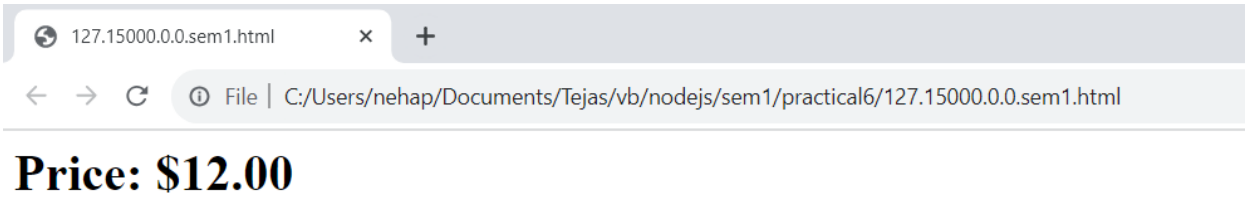
Practical 7

Aim - Create an application using Filters.

Code -

```
<html>
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
 <body>
    <div ng-app="myApp" ng-controller="costCtrl">
        <h1>Price: {{ price | currency }}</h1>
    </div>
    <script>
        var app = angular.module('myApp', []);
        app.controller('costCtrl', function($scope) {
        $scope.price = 12;
        });
    </script>
 </body>
</html>
```

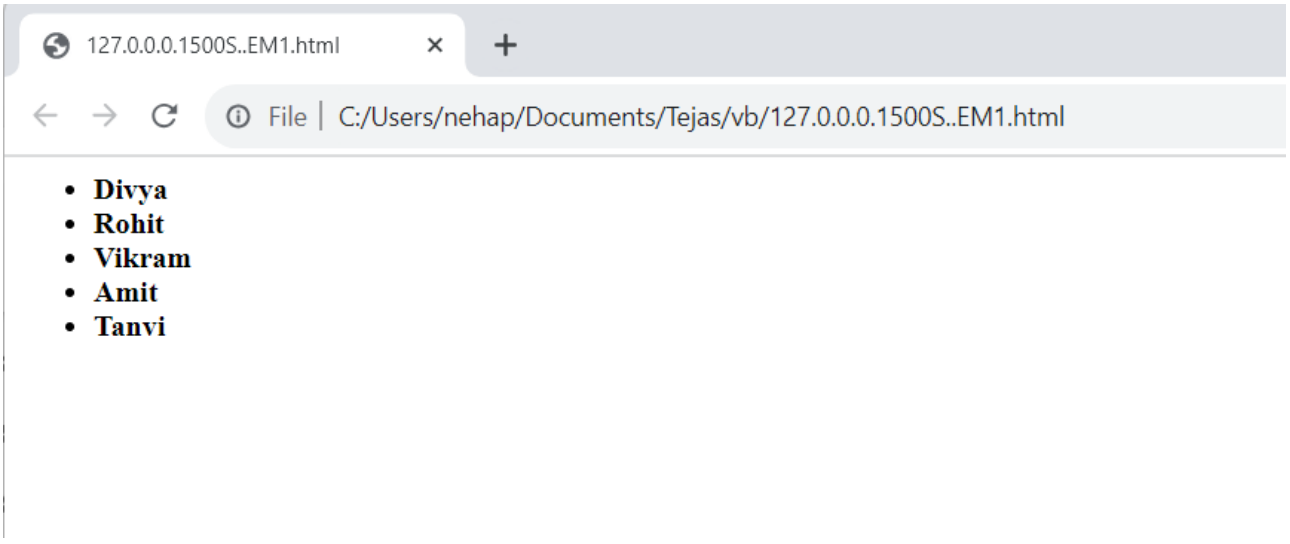
Output -



Code:-

```
<html>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <body>
    <div ng-app="myApp" ng-controller="namesCtrl">
      <ul>
        <strong>
          <li ng-repeat="x in names | filter : 'i'">{{ x }}</li>
        </strong>
      </ul>
    </div>
    <script>
      angular.module("myApp", []).controller("namesCtrl", function ($scope) {
        $scope.names = [
          "Divya",
          "Rahul",
          "Pooja",
          "Rohit",
          "Shreya",
          "Vikram",
          "Neha",
          "Arjun",
          "Kavya",
          "Amit",
          "Tanvi",
          "Harsh",
        ];
      });
    </script>
  </body>
</html>
```

Output -

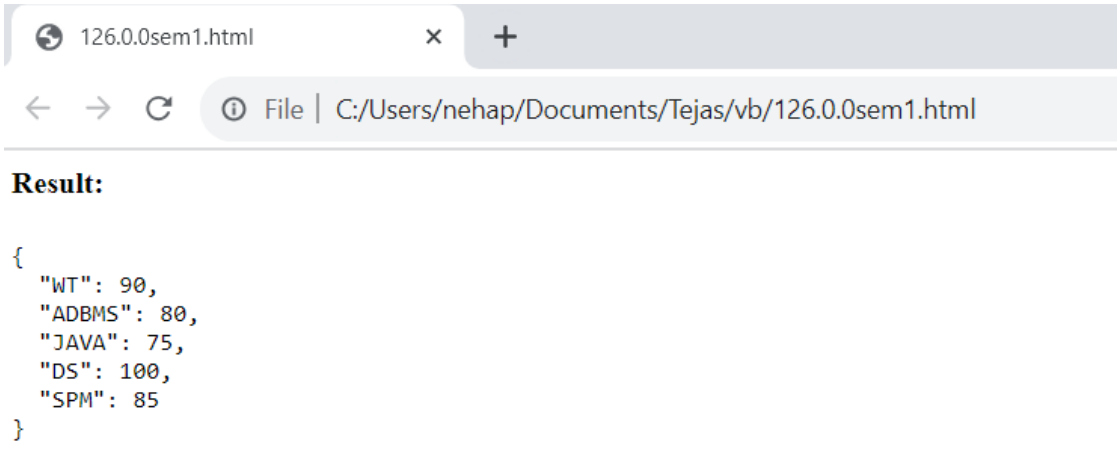


Code:-

Code -

```
<html>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <body>
    <div ng-app="result" ng-controller="resultCtrl">
        <h4>Result:</h4>
        <pre>{{marks | json}}</pre>
    </div>
    <script>
        var app = angular.module('result', []);
        app.controller('resultCtrl', function ($scope) {
            $scope.marks = {
                "WT": 90,
                "ADBMS": 80,
                "JAVA": 75,
                "DS": 100,
                "SPM": 85
            };
        });
    </script>
  </body>
</html>
```

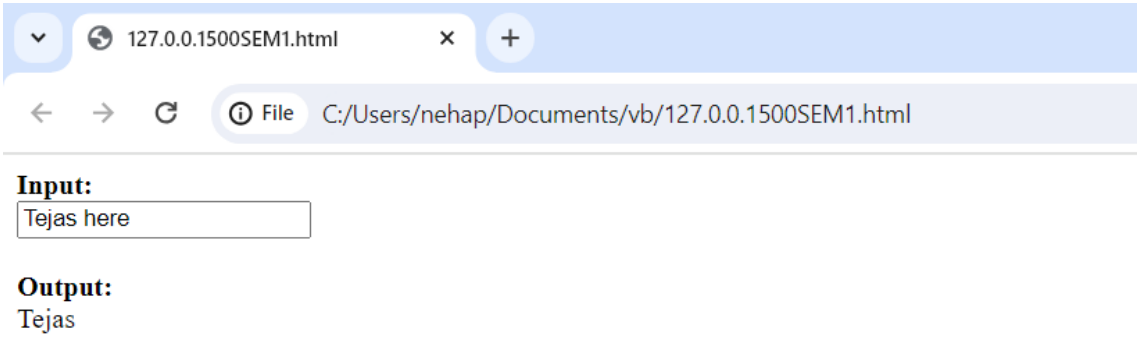
Output -



Code:-

```
<html>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <body>
    <div ng-app="myApp" ng-controller="myCtrl">
      <strong>Input:</strong>
      <br />
      <input type="text" ng-model="string" />
      <br />
      <br />
      <strong>Output:</strong>
      <br />
      {{string|limitTo:6}}
    </div>
    <script>
      var app = angular.module("myApp", []);
      app.controller("myCtrl", function ($scope) {
        $scope.string = "";
      });
    </script>
  </body>
</html>
```

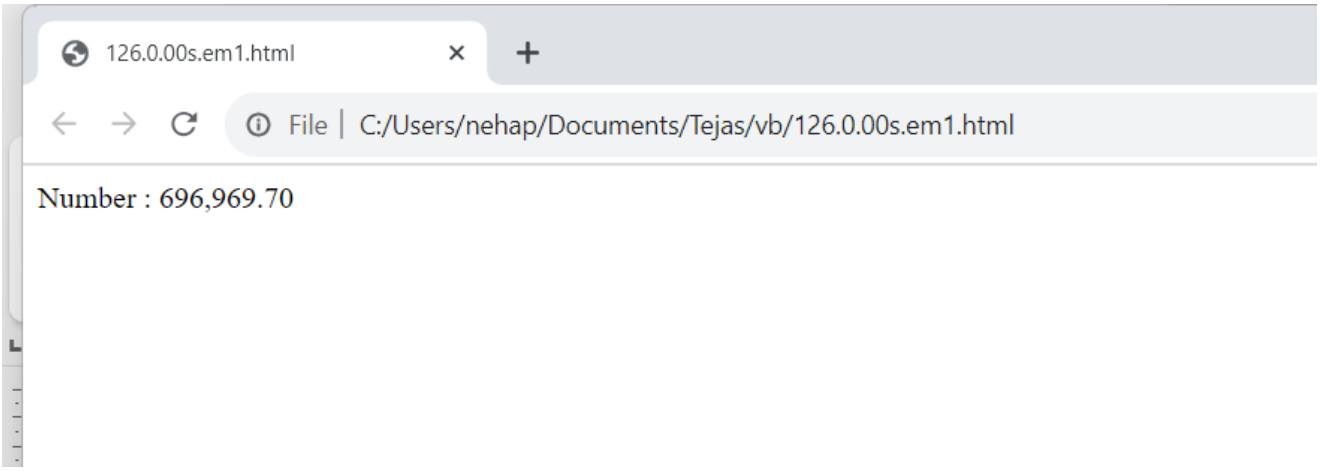
Output -



Code:-

```
<html>
  <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <body>
    <div ng-app="myApp" ng-controller="numberCntrl">
      <p>Number : {{ value | number : 2}}</p>
    </div>
    <script>
      var app = angular.module("myApp", []);
      app.controller("numberCntrl", function ($scope) {
        $scope.value = 696969.6969;
      });
    </script>
  </body>
</html>
```

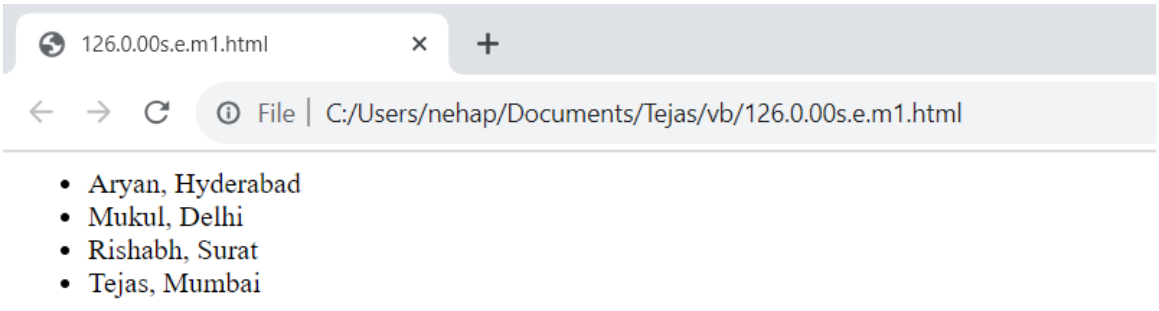
Output -



Code:-

```html
<html>
  <script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <body>
    <div ng-app="myApp" ng-controller="orderCtrl">
        <ul>
        <li ng-repeat= "x in customers | orderBy : 'name'">
              {{x.name + ", " + x.city}}
            </li>
        </ul>
    </div>
    <script>
        var app = angular.module('myApp', []);
        app.controller('orderCtrl', function($scope) {
            $scope.customers = [{
                "name": "Tejas",
                "city": "Mumbai"
            }, {
                "name": "Rishabh",
                "city": "Surat"
            }, {
                "name": "Mukul",
                "city": "Delhi"
            }, {
                "name": "Aryan",
                "city": "Hyderabad"
            }, ];
        });
    </script>
  </body>
</html>
```

Output:-



Practical 8

Aim - Create an application to demonstrate Directive.

Theory - AngularJS introduces the concept of Directives, allowing the extension of HTML with additional attributes. It provides a set of built-in directives that enhance application functionality. Furthermore, AngularJS enables users to define their own directives. These directives are characterized by the prefix "ng-" in extended HTML attributes.

The ng-app directive serves to initialize an AngularJS application, while ng-init is used to initialize application data. The ng-model directive facilitates the binding of HTML control values (such as input, select, and textarea) to application data.

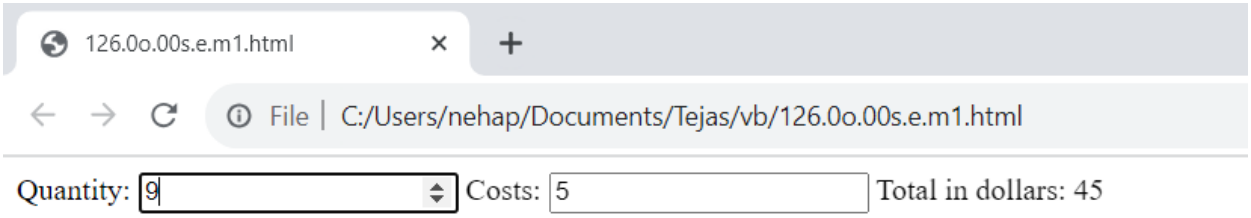
Data binding using ng-model is exemplified through the following code snippets:

Example 1:

Code -

```
<html>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <body>
    <div ng-app="" ng-init="quantity=1;price=5">
      Quantity: <input type="number" ng-model="quantity">
      Costs: <input type="number" ng-model="price">
      Total in dollars: {{ quantity * price }}
    </div>
  </body>
</html>
```

Output:-

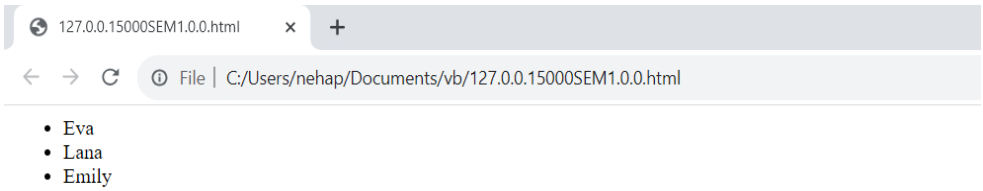


Example 2:

Code -

```
<html>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angul
ar.min.js"></script>
 <body>
   <div ng-app="" ng-init="names=['Eva','Lana','Emily']">
     <ul>
       <li ng-repeat="x in names">{{ x }}</li>
     </ul>
   </div>
 </body>
</html>
```

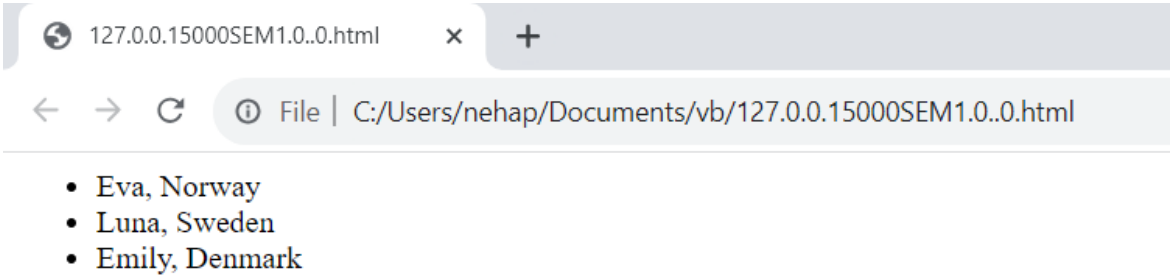
Output -



Code -

```
<html>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angul
ar.min.js"></script>
 <body>
   <div
     ng-app=""
     ng-init="names=[
     {name:'Eva',country:'Norway'},
     {name:'Luna',country:'Sweden'},
     {name:'Emily',country:'Denmark'}]"
   >
     <ul>
       <li ng-repeat="x in names">{{ x.name + ', ' + x.country
}}</li>
     </ul>
   </div>
 </body>
</html>
```

Output -



Creating New Directives:

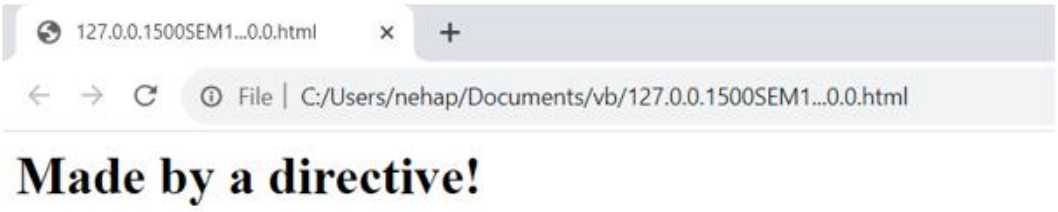
1. AngularJS provides the capability to create custom directives in addition to the built-in ones.
2. The creation of new directives involves using the `.directive` function.
3. When invoking a new directive, an HTML element with the same tag name as the directive should be created.
4. While naming a directive, camel case is used (e.g, `w3TestDirective`), but when invoking it, a hyphen-separated name is employed (e.g., `w3-test-directive`).

Example:

Code -

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

Output -



Invoking Directives:

1. Directives can be invoked using different methods:
    - Element: `` `<w3-test-directive></w3-test-directive>` ``
    - Attribute: `` `<div w3-test-directive></div>` ``
    - Class: `` `<div class="w3-test-directive"></div>` ``
    - Comment: `` `<!-- directive: w3-test-directive -->` ``

Mouse Events Directives:

1. **Angular ng-mousedown Directive:**
    - AngularJS `ng-mousedown` directive handles actions when a mouse button is clicked on an HTML element.
    - It does not override the original `onmousedown` event, and both will be executed.

Example:

```
<html>

 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/an
gular.min.js"></script>

 <body ng-app="myApp">
```

```
   <body ng-app="">

     <div ng-mousedown="count = count + 1"
ng-init="count=0">Click me!</div>

     <h1>{{count}}</h1>

     <p>

       This example increases the value of the variable
"count" every time a

       mouse button is clicked on the DIV element.

     </p>

   </body>

 </body>

</html>
```

2. **AngularJS ng-mouseenter Directive:**

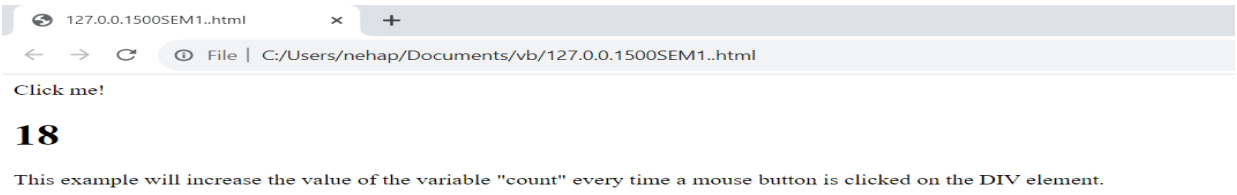
The `ng-mouseenter` directive instructs AngularJS on the actions to take when the mouse cursor enters a specific HTML element. Similar to `ng-mousedown`, it does not override the element's original `onmouseenter` event, and both events will be executed.

Code -

```
<!DOCTYPE html>

<html>
```

```
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/an
gular.min.js"></script>

 <body ng-app="">

   <div ng-mouseenter="count = count + 1"
ng-init="count=0">Hover over me!</div>

   <h1>{{count}}</h1>

   <p>This example increases the value of the variable
"count" every time the mouse enters the DIV element.</p>

 </body>

</html>
```

Output -



Other Mouse Events Directives

3. **AngularJS ng-mouseleave Directive**
4. **AngularJS ng-mousemove Directive**
5. **AngularJS ng-mouseover Directive**
6. **AngularJS ng-mouseup Directive**

Keyboard Events Directives:

1. **AngularJS ng-keydown Directive:**

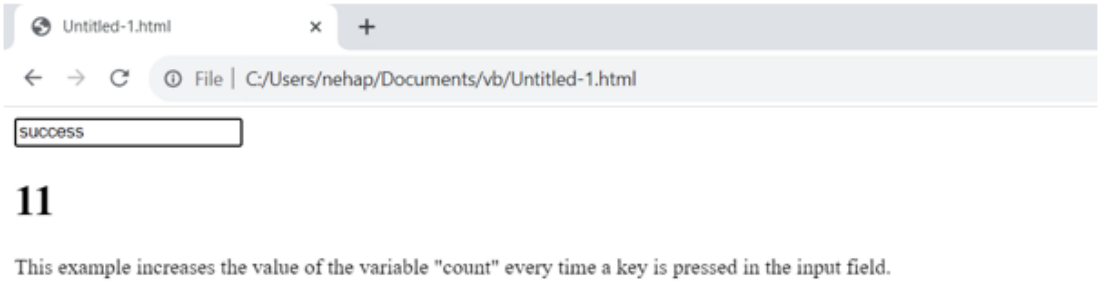
Theory: The ng-keydown directive in AngularJS defines the actions to take when the keyboard is used on a specific HTML element. Like other directives, it does not override the element's original onkeydown event, and both events will be executed. The order of a keystroke is:

1. Keydown
2. Keypress
3. Keyup

Code -

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/an
gular.min.js"></script>
<body ng-app="">
 <input ng-keydown="count = count + 1" ng-init="count=0" />
 <h1>{{count}}</h1>
 <p>This example increases the value of the variable "count"
every time a key is pressed in the input field.</p>
</body>
</html>
```

Output -



2. **AngularJS ng-keypress Directive**
3. **AngularJS ng-keyup Directive**

Practical 9

Aim - Demonstrate Controllers in Angular.js through an application.

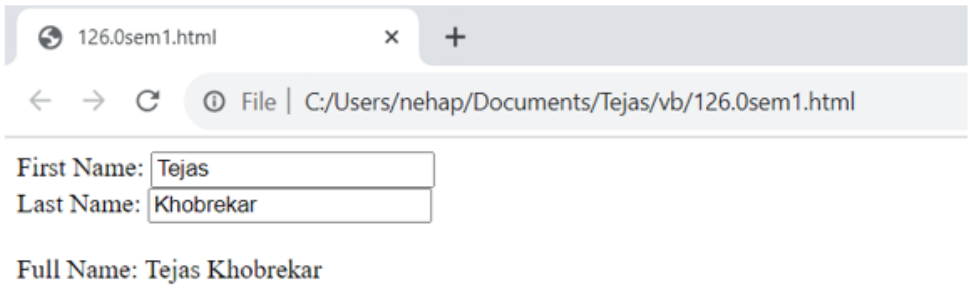
Theory - AngularJS controllers manage the data within AngularJS applications and are essentially regular JavaScript Objects. These controllers play a pivotal role in controlling AngularJS applications. The `ng-controller` directive is used to define the application controller. Controllers, in AngularJS, are JavaScript Objects created through standard JavaScript object constructors.

Code -

```
<!DOCTYPE html>

<html>

<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" ng-model="firstName"><br>

Last Name: <input type="text" ng-model="lastName"><br>

<br>

Full Name: {{firstName + " " + lastName}}

</div>

<script>
```

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope)
{

$scope.firstName="Tejas";

$scope.lastName = "Khobrekar";

});
</script>
</body>
</html>
```

Output -

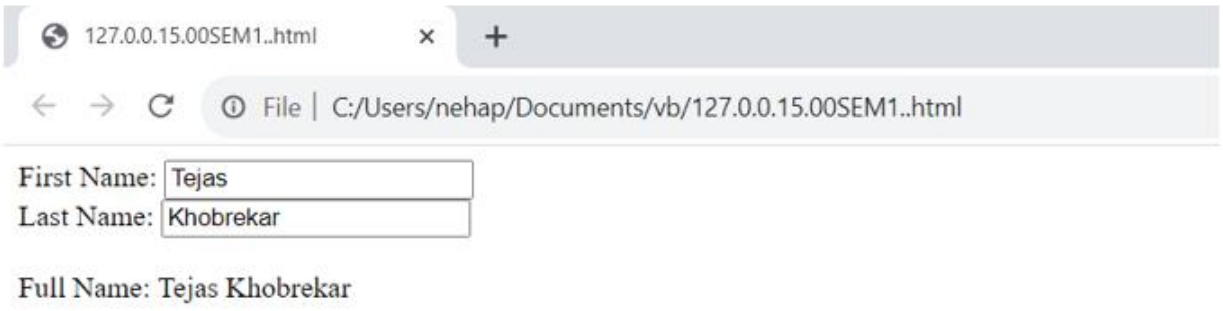


The AngularJS application is defined by `ng-app="myApp"`, running inside the `<div>`. The `ng-controller="myCtrl"` attribute is an AngularJS directive defining a controller. The `myCtrl` function is a JavaScript function invoked by AngularJS with a `$scope` object. In AngularJS, `$scope` is the application object, controlling application variables and functions. The controller creates two properties (`firstName` and `lastName`) in the scope, and `ng-model` directives bind the input fields to these controller properties.

Controller Methods:

```
<!DOCTYPE html>

<html>

<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" ng-model="firstName"><br>

Last Name: <input type="text" ng-model="lastName"><br> <br>

Full Name: {{fullName()}}
</div>


<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {
$scope.firstName = "Tejas";

$scope.lastName ="Khobrekar";

$scope.fullName = function() {

return $scope.firstName + " "+ $scope.lastName;
};
});

</script>

</body>

</html>
```

Output -


127.0.0.15.00SEM1..html
File | C:/Users/nehap/Documents/vb/127.0.0.15.00SEM1..html
First Name: Tejas
Last Name: Khobrekar
Full Name: Tejas Khobrekar

Practical 10

Aim - Demonstrate features of Angular.js forms with a program.

Theory - Forms in AngularJS provide data-binding and validation for input controls, including various types like text, email, number, and URL. The form and input fields have different states and classes, indicating whether they are touched, modified, valid, or invalid.

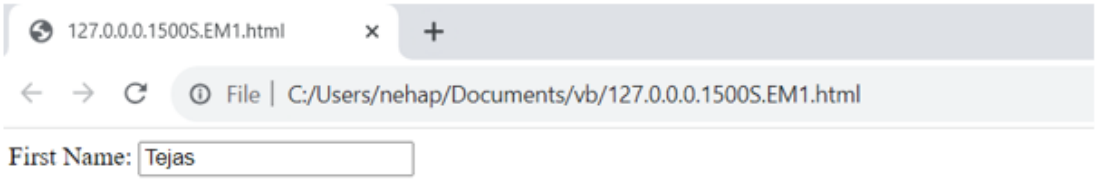
Input Controls:

Input controls include HTML input elements like text, select, button, and textarea. Below is an example:

Code -

```
<!DOCTYPE html>

<html lang="en">

<script

src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="formCtrl">

<form>

First Name: <input type="text" ng-model="firstname">

</form>

</div>
<script>
var app = angular.module('myApp',[]);
 app.controller('formCtrl', function ($scope){
    $scope.firstname="Tejas";
 });
</script>

</body>

</html>
```

Output -



The initial value for the first name is set to “Tejas”

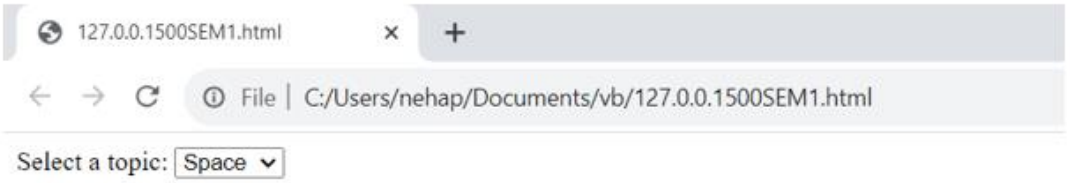
Selectbox: Bind select boxes to your application with the `ng-model` directive. Below is an example:

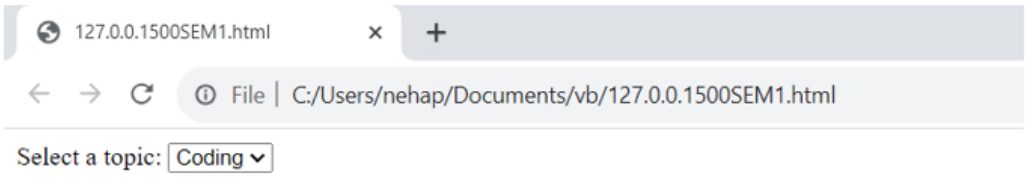
Code -

```
<!DOCTYPE html>

<html>

<script>

src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<body ng-app="">

<form>

Select a topic:

<select ng-model="myVar">

<option value=""></option>

<option value="space">Space</option>

<option value="coding">Coding</option>
option value "travel">Travel option>

</select>
```

```
</form>

<div ng-switch-"myVar">

<div ng-switch-when-"space">

<h1>Space</h1>

<p>Welcome to the wonders of space.</p>

</div>

<div ng-switch-when-"coding">

<h1>Coding</h1>

<p>Explore the world of programming.</p>

</div>

<div ng-switch-when-"travel">

<h1>Travel</h1>

<p>Discover exciting places around the globe.</p>

</div>

</div>

<p>The 'ng-switch directive hides and shows HTML sections depending on the value of the dropdown list.</p>

</body>

</html>
```

Output -





The output will dynamically display content based on the selected option.

Angular Form Validation:

To validate whether the field is empty, HTML5 uses the required attribute. The syntax is as follows:

```
<input type="text" required />
```

AngularJS provides flexibility with the ng-required directive, allowing you to dynamically set whether the input field should have a value:

```
<input type="text" ng-required="true" />
```

2. **Minimum Length:**

The ng-minlength directive validates the minimum length of the input value:

```
<input type="text" ng-minlength="10" />
```

3. **Maximum Length:**

The ng-maxlength directive validates the maximum length of the input value:

```
<input type="text" ng-maxlength="20" />
```

4. **Pattern:**

The ng-pattern directive ensures that an input matches a regex pattern:

```
<input type="text" ng-pattern="[a-zA-Z]" />
```

5. **Email:**

Setting the input type to email ensures the input field contains a valid email ID:

```
<input type="email" name="email"
ng-model="user.email" />
```

6. **Number:**

Setting the input type to number ensures the input field only accepts numeric values:

```
<input type="number" name="age" ng-model="user.age"
/>
```

7. **URL:**

Setting the input type to url ensures the input field contains a valid URL:

```
<input type="url" name="homepage"
ng-model="user.url" />
```

Form State and Input State: AngularJS constantly updates the state of both the form and input fields. Input fields have the following states:

- $untouched: The field has not been touched yet
- $touched: The field has been touched
- $pristine: The field has not been modified yet
- $dirty: The field has been modified
- $invalid: The field content is not valid
- $valid: The field content is valid

CSS Classes: AngularJS adds CSS classes to forms and input fields based on their states. Classes include ng-untouched, ng-touched, ng-pristine, ng-dirty, ng-valid, ng-invalid, ng-valid-key, and ng-invalid-key.

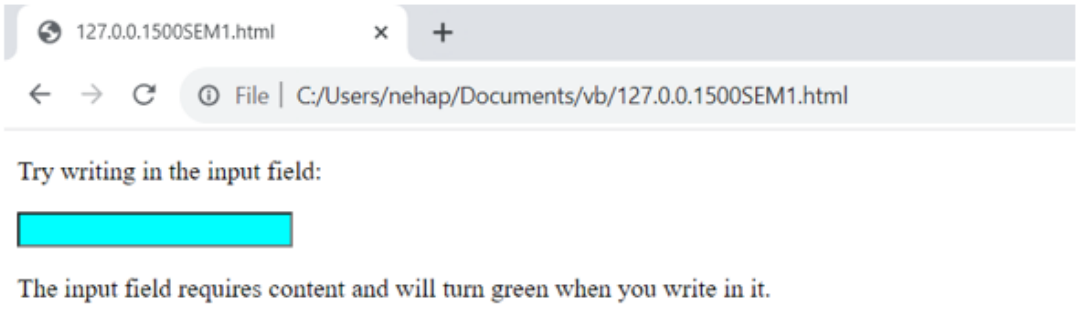
Example 1:

Code -

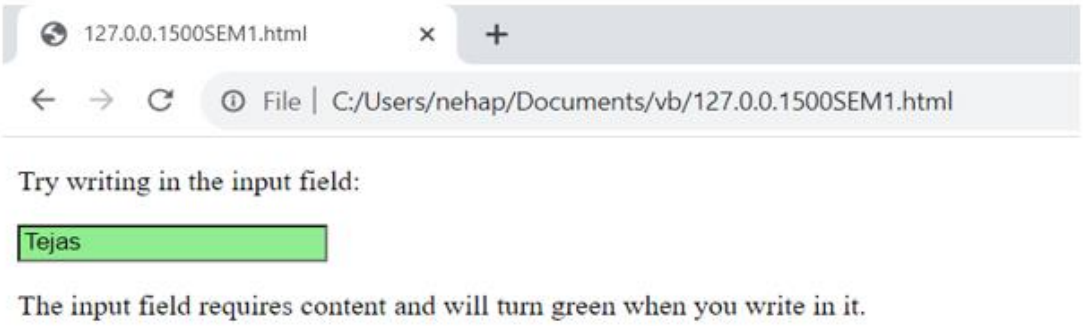
```
<!DOCTYPE html>

<html>

<script

src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<style>

input.ng-invalid { background-color: cyan;}

input.ng-valid { background-color: lightgreen;
}
</style>

<body ng-app = "">
    p>Try writing in the input field:</p>

<form name="myForm">

<input name="myName" ng-model="myName" required>

</form>

<p>The input field requires content and will turn green when you write in it.</p>

</body>

</html>
```

Output -

Before typing anything in the field



After typing in the field -



Example 2:

Code -

AngularForm.html

```
<html>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
  <style>
    input.ng-invalid {
      background-color: rgb(128, 201, 240);
    }
    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:
                                         &
                <input
                  type="text"
                  name="name"0
                  style="padding: 5px"
                  ng-model="fullname"
                />
              </b>
            </h3>
```

```
<h3>
  <b
    >Appartment Number:
                   
    <input
      type="text"
      name="appartmentnumber"
      style="padding: 5px"
      ng-model="appartmentnumber"
      required
    />
  </b>
</h3>
<h3>
  <b
    >Electricity Meter Number:        
    <input
      type="text"
      name="numelectricity"
      style="padding: 5px"
      ng-model="meternumber"
    />
  </b>
</h3>
```

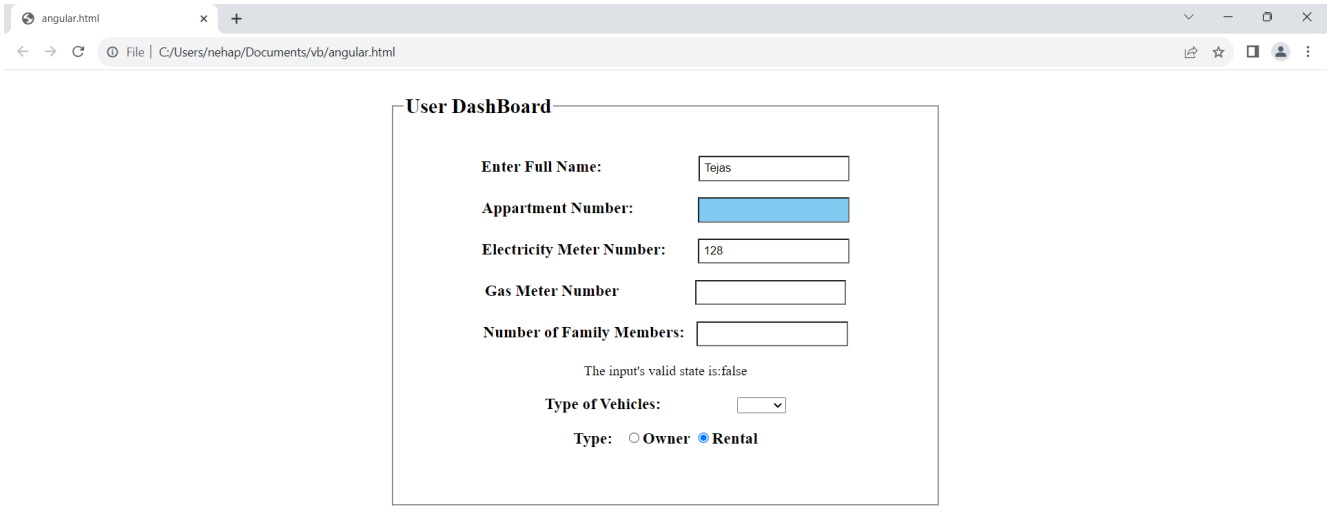
```
          <h3>
            <b
              >Gas Meter Number                       
              <input
                type="text"
                name="numgasmeter"
                style="padding: 5px"
                ng-model="gasnumber"
              />
            </b>
          </h3>
          <h3>
            <b


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

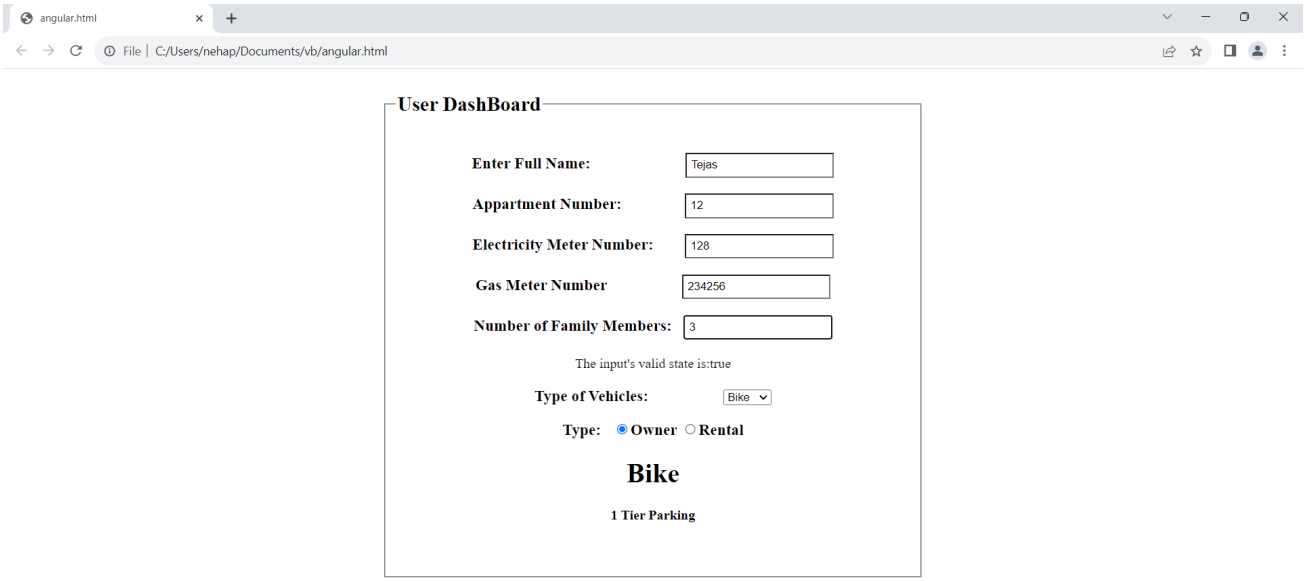
```
            <div ng-switch-when="Truck">
              <h1>Truck</h1>
              <p><b>4 Tier Parking</b></p>
            </div>
          </div>
          <br /><br />
        </fieldset>
      </form>
    </center>
    <script>
      var app = angular.module("myApp", []);
      app.controller("formCtrl", function ($scope) {
        $scope.fullname = "Tejas";
        $scope.meternumber = "128";
      });
    </script>
  </body>
</html>
```

Output -

Before Input:



After Input:


angular.html
File | C:/Users/nehap/Documents/vb/angular.html
User DashBoard
Enter Full Name:
Tejas
Appartment Number:
12
Electricity Meter Number:
128
Gas Meter Number
234256
Number of Family Members:
3
The input's valid state is:true
Type of Vehicles:
Bike
Type:
Owner
Rental
Bike
1 Tier Parking

Practical 11

Aim: 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>
```

```
    <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 -

ATHLETICS FEDERATION OF INDIA

Cricket Football Hockey

Cricket

Cricket Where Skills Create Music!

Message From Cricket Federation

ATHLETICS FEDERATION OF INDIA

Cricket Football Hockey

Football

Game of Warriors
Apply now!

Message From Football Department

ATHLETICS FEDERATION OF INDIA

Cricket Football Hockey

Hockey

Game of Legends
Apply now!

Message From Hockey Department