Experiment No. 1

AIM: What is node js?

Objective: Explain node JS

Step 1: Install Node.js and NPM on

Windows Step 2: Install Node.js and

NPM from Browser Step 3: Verify

Installation

```
:\Users\ANKUSH>node -v
18.13.0

:\Users\ANKUSH>npm -v
.19.3

:\Users\ANKUSH>


:\Users\ANKUSH>node
elcome to Node.js v18.13.0.
ype ".help" for more information.
 1+3

 1+(2*3)-4

 85*8/4+8-2
76
```

1 console.log([data][, ...])

2 console.info([data][, ...])

3 console.error([data][, ...])

4 console.warn([data][, ...])

5 console.dir(obj[, options])

6 console.time(label)

7 console.timeEnd(label)

8 console.trace(message[, ...])

9 console.assert(value[, message][, ...])

```
var counter = 10;
console.log("Counter: %d", counter);

console.time("Getting data");
//
// Do some processing here...
//
console.timeEnd('Getting data');

console.info("Program Ended")
```

```
My First Program
Counter: 10
Getting data: 0.234ms
Program Ended
```

EXPERIMENT No. 2

AIM: To demonstrate the use of Standard callback pattern

Objective: Node js callback pattern function callback

Theory: Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

```
Get Started   JS First.js   JS Second.js ×   check.txt
F: > MumbaiUniversity > Nodejs > Practice > JS Second.js > ...
var fs = require("fs"); var data = fs.readFileSync('check.txt');

console.log(data.toString()); console.log("Program Ended");
```

```
Node.js v18.13.0
PS F:\MumbaiUniversity\Nodejs\Practice> node Second.js
Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task.
 Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.
Program Ended
```

AIM: To demonstrate the event emitter pattern

Objective: Explanation of event emitter pattern with programme.

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

// listener #1
var listner1 = function listner1() {
   console.log('listner1 executed.');
}

// listener #2
var listner2 = function listner2() {
   console.log('listner2 executed.');
}

// Bind the connection event with the listner1 function
eventEmitter.addListener('connection', listner1);

// Bind the connection event with the listner2 function
eventEmitter.on('connection', listner2);

var eventListeners = require('events').EventEmitter.listenerCount
   (eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");

// Fire the connection event
eventEmitter.emit('connection');

// Remove the binding of listner1 function
eventEmitter.removeListener('connection', listner1);
console.log("Listner1 will not listen now.");

// Fire the connection event
eventEmitter.emit('connection');

eventListeners = require('events').EventEmitter.listenerCount(eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");

console.log("Program Ended.");
```

```
PS F:\MumbaiUniversity\Nodejs\Practice> node Second.js
2 Listner(s) listening to connection event
listner1 executed.
listner2 executed.
Listner1 will not listen now.
listner2 executed.
1 Listner(s) listening to connection event
Program Ended.
```

AIM: To demonstrate the use of defer execution of a function

Objective: Implement defer execution in node JS function

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

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

//Subscribe for FirstEvent
 em.on('FirstEvent', function (data) {     console.log('First subscriber: ' + data); });

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

```
PS F:\MumbaiUniversity\Nodejs\Practice> node Second.js
First subscriber: This is my first Node.js event emitter example.
PS F:\MumbaiUniversity\Nodejs\Practice>
```

Aim: To demonstrate the use stop execution of a function

Objective: Using process.exit() method stop function execution

```
Error: Something went wrong
    at Object.<anonymous> (F:\MumbaiUniversity\Nodejs\Practice\Second.js:81:7)
    at Module._compile (node:internal/modules/cjs/loader:1218:14)
    at Module._extensions..js (node:internal/modules/cjs/loader:1272:10)
    at Module.load (node:internal/modules/cjs/loader:1081:32)
    at Module._load (node:internal/modules/cjs/loader:922:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:81:12)
    at node:internal/main/run_main_module:23:47

Node.js v18.13.0
```

```
var thisIsTrue = false;
exports.test = function(request, response, cb){
if (thisIsTrue)
{
    response.send('All is good!');
    cb(null, response)
}
else
{
  response.send('ERROR! ERROR!');
  return cb("THIS ISN'T TRUE!");
}
  console.log('I do not want this to happen. If there is an error.');
}
throw new Error('Something went wrong');
```

AIM: To demonstrate the use Schedule and repetitive execution

```
function sayHi()
{
 alert('Hello');
//  println('Hello');
}

//2
let timeout;

function myFunction() {
  timeout = setTimeout(alertFunc, 3000);
}

function alertFunc() {
  alert("Hello!");
}
```

Objective: Using setTimeout & setInterval Schedule and repetitive execution

```
PS F:\MumbaiUniversity\Nodejs\Practice> node  Second.js
PS F:\MumbaiUniversity\Nodejs\Practice>
```

```
console.log('before immediate');

setImmediate((arg) => {
  console.log(`executing immediate: ${arg}`);
}, 'so immediate');

console.log('after immediate');
```

```
PS F:\MumbaiUniversity\Nodejs\Practice> node Second.js
before immediate
after immediate
executing immediate: so immediate
```

```
let timerId = setInterval(() => alert('tick'), 2000);

// after 5 seconds stop
setTimeout(() => { clearInterval(timerId); alert('stop'); }, 5000);
3
```


```
let timerId = setInterval(() => alert('tick'), 2000);

timerId = setTimeout(function tick() {   alert('tick');   timerId = setTimeout(tick,
2000); // (*) }, 2000);
```

The setTimeout above schedules the next call right at the end of the current one (*).

The nested setTimeout is a more flexible method than setInterval. This way the next call may be scheduled differently, depending on the results of the current one.

AIM: To demonstrate the use Block escape event loop

Objective: Using block loop method run node js function

Theory: Now that we have a healthy refresh on how threads work, we can finally tackle the Node.js event loop logic. By reading this, you will understand the reason behind the previous explanation, and every piece will go at the right spot by itself.

```
setTimeout(() =>
 {
  const delay = Date.now() - timeoutScheduled;

  console.log(`${delay}ms have passed since I was scheduled`);
}, 100);

// do someAsyncOperation which takes 95 ms to complete
someAsyncOperation(() => {   const startCallback = Date.now();

  // do something that will take 10ms...
  while (Date.now() - startCallback < 10) {     // do nothing
   } });
```

EXPERIMENT No. 3

AIM: To demonstrate the Fs module file paths

Objective: Explanation of Fs module file paths

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

var stats = fs.statSync("F:/MumbaiUniversity/Nodejs/Practice");

console.log('is file ? ' + stats.isFile());

var stats = fs.statSync("F:/MumbaiUniversity/Nodejs/Practice/Second.js");

console.log('is directory ? ' + stats.isDirectory());
```

```
PS F:\MumbaiUniversity\Nodejs\Practice> node third.js
is file ? false
is directory ? false
PS F:\MumbaiUniversity\Nodejs\Practice>
```

AIM: To demonstrate the how to read, write, & close file

Objective: Explantion of the how to read, write, & close file in node.js

```
var fs = require("fs");
fs.readFile("temp.txt", function(err, buf)
{
    console.log(buf.toString());
});
```

```
PS F:\MumbaiUniversity\Nodejs\Practice> node Third.js
AIM: To demonstrate the how to read, write, & close file

Objective: Explantion of the how to read, write, & close file in node.js

Theory Reading From Files: Being able to read from files on your local file system can be hugely useful and there are a number of different things you can build on top of this. A log reader, importing
 information from spreadsheets and xml files or whatever you can think of, being able to read from files is hugely useful the file path is File, otherwise returns false. The stats.isDirectory() method
 returns true if file path is Directory, otherwise returns false
PS F:\MumbaiUniversity\Nodejs\Practice>
```

```
var fs = require("fs");
fs.readFile("temp.txt", function(err, buf)
{
    console.log(buf.toString());
});

fs.readFile("temp.txt", function(err, buf)
{
    console.log(buf);
});
```

```
PS F:\MumbaiUniversity\Nodejs\Practice> node Third.js
AIM: To demonstrate the how to read, write, & close file

Objective: Explantion of the how to read, write, & close file in node.js

Theory Reading From Files: Being able to read from files on your local file system can be hugely useful and there are a number of different things you can build on top of this. A log reader, importing
 information from spreadsheets and xml files or whatever you can think of, being able to read from files is hugely useful the file path is File, otherwise returns false. The stats.isDirectory() method
 returns true if file path is Directory, otherwise returns false
<Buffer 41 49 4d 3a 20 20 54 6f 20 64 65 6d 6f 6e 73 74 72 61 74 65 20 74 68 65 20 68 6f 77 20 74 6f 20 72 65 61 64 2c 20 77 72 69 74 65 2c 20 26 20 63 6c 6f ... 558 more bytes>
```

AIM: Demonstrate how to read data in SQL using node js

Objective: Explanation how to read data in SQL using node js

Explanation: The steps for querying data in the MySQL database from a node.js application are as follows: 1. Establish a connection to the MySQL database server. 2. Execute a SELECT statement and process the result set. 3. Close the database connection.

```
let mssql = require('mssql');
//let config = require('./node_modules/config');
let config = require('./config.js');
let connection = mssql.({host: "1433",
user: "ankush",
password: "123456",
database: "NodeTest"});
//sql.connect(config).then(function(mssql)

let sql = `SELECT * FROM tblSample`;
connection.query(sql, (error, results, fields) =>
{
      if (error)
     {
          return console.error(error.message);
     }
      console.log(results); });
      connection.end();
```

EXPERIMENT No. 4

AIM: Write a program to display your name with welcome note: HELLO

Objective: Understand basic AngularJS components such as Modules, Directives, Expressions, Controllers, Services and Filters Understand the basic design of AngularJS Build AngularJS forms and bind data to objects

THEORY:

Before creating actual Hello World ! application using AngularJS, let us see the parts of a AngularJS application. An AngularJS application consists of following three important parts − ng-app: This directive defines and links an AngularJS application to HTML. ng-model: This directive binds the values of AngularJS application data to HTML input controls. ng-bind: This directive binds the AngularJS Application data to HTML tags.

F:\MumbaiUniversity\Nodejs\Practice\AngularSeconds.html • - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

SinglePageOne.html | AngularSeconds.html

```
<hr><!doctype html>
<html ng-app>   <head>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js">
  </script>
</head>
<body>
  <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <h1>Hello {{yourName}}!</h1>
  </div>
</body>
</html>
```

Name: Ankush

Hello Ankush!

EXPERIMENT No. 5

AIM: To create a real AngularJS Application for shopping Cart

Objective: Use some of the AngularJS features to make a shopping list, where user can add or remove items:

Shopping List

 Milk×

Bread×

 Cheese×

THEORY:

A shopping cart is similar to original grocery shopping cart; it means that on a website that sells products or services online, the shopping cart is a common metaphor that acts as online store‟s catalog and ordering process. It is a graphical representation of a supermarket on a vendor‟s website that keeps a list of items a customer has picked up from the online store.

Shopping cart is an infrastructure that allows customers to review what they have selected, make necessary modifications such as adding or deleting products and purchase the merchandise. Customer checks off the products that are being ordered and when finished ordering, that proceeds to a page where the total order is confirmed and placed. Also, customers will enter their shipping tax information at the checkout. Shopping cart allows a website to build a catalog of products and integrate it into the website pages.

Shopping cart is important infrastructure to have smooth ecommerce transition. The shopping cart describes specialized content management system that includes,

website wizards

provides portal for catalog, order and customer management

renders product data, product categories

merchant tools shopping features

payment options

shipping and tax information

passes transactional data to payment gateway

statistics and security

F:\MumbaiUniversity\Nodejs\Practice\FourThree.html - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

SinglePageOne.html | AngularSeconds.html | FourThree.html

```html
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
<body>

<script>
var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
    $scope.products = ["Milk", "Bread", "Cheese"];
    $scope.addItem = function () {
        $scope.errortext = "";
        if (!$scope.addMe) {return;}
        if ($scope.products.indexOf($scope.addMe) == -1) {
            $scope.products.push($scope.addMe);
        } else {
            $scope.errortext = "The item is already in your shopping list.";
        }
    }
    $scope.removeItem = function (x) {
        $scope.errortext = "";
        $scope.products.splice(x, 1);
    }
});
</script>

<div ng-app="myShoppingList" ng-cloak ng-controller="myCtrl" class="w3-card-2 w3-margin" style="max-width:400px;">
  <header class="w3-container w3-light-grey w3-padding-16">
    <h3>My Shopping List</h3>
  </header>
  <ul class="w3-ul">
    <li ng-repeat="x in products" class="w3-padding-16">{{x}}<span ng-click="removeItem($index)" style="cursor:pointer;" class="w3-right w3-margin-right">×</
    span></li>
  </ul>
  <div class="w3-container w3-light-grey w3-padding-16">
    <div class="w3-row w3-margin-top">
      <div class="w3-col s10">
        <input placeholder="Add shopping items here" ng-model="addMe" class="w3-input w3-border w3-padding">
      </div>
      <div class="w3-col s2">
```

My Shopping List

Milk ×

Bread ×

Cheese ×

Add shopping items here | Add

EXPERIMENT NO. 6

AIM: Create Simple User Registration Form in AngularJS OBJECTIVE:

User Registration is very basic and common feature in modern web application. It is one of the basic and most important features for a web application that is used to authenticate or restrict unauthorized access to member only areas and features in a web application. The user is required to register an account using the registration form provided on the application so the username and password is created in order to login in the application.

Index.html

F:\MumbaiUniversity\Nodejs\Practice\FiveTwo.html - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

SinglePageOne.html | AngularSeconds.html | FourThree.html | fiveTwo | FiveTwo.html | app.js

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>User Registraion Form - W3Adda</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js"></script>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
<script src="app.js"></script>
</head>
<body>
<div ng-app = "myApp" class = "container" style="width:550px">
<div style="text-align:center;">
<h3><b>User Registraion Form - W3Adda</b></h3>
</div>
<div ng-controller = "RegisterController">m
<div align="right">
<a href="#" ng-click="searchUser()">{{title}}</a>
</div>
<form role = "form" class="well" ng-hide="ifSearchUser">
<div class = "form-group">
<label for = "name"> Name:  </label>
<input type = "text" id = "name" class = "form-control" placeholder = "Enter Name " ng-model = "newuser.name">
</div>
<div class = "form-group">
<label for = "email"> Email:  </label>
<input type = "email" id = "email" class = "form-control" placeholder = "Enter Email " ng-model = "newuser.email">
</div>
<div class = "form-group">
<label for = "password"> Password:  </label>
<input type = "password" id = "password" class = "form-control" placeholder = "Enter Password " ng-model = "newuser.password">
</div>
<div class = "form-group">
<label for = "phone"> Phone:  </label>
```

```html
<div><h4><b>Registered Users</b></h4>
<table ng-if="users.length" class = "table table-striped table-bordered table-hover">
<thead>
<tr class = "info">
<th>Name</th>
<th>Email</th>
<th>Phone</th>
<th ng-if="!ifSearchUser">Action</th>
</tr>
</thead>
<tbody>
<tr ng-repeat = "user in users">
<td>{{ user.name }}</td>
<td>{{ user.email }}</td>
<td>{{ user.phone }}</td>
<td ng-if="!ifSearchUser">
<a href="#" ng-click="edit(user.id)" role = "button" class = "btn btn-info">edit</a>  
<a href="#" ng-click="delete(user.id)" role = "button" class = "btn btn-danger">delete</a>
</td>
</tr>
</tbody>
</table>
<div ng-hide="users.length > 0">No Users Found</div>
</div>
</div>
</div>
</body>
</html>
```

App.js

F:\MumbaiUniversity\Nodejs\Practice\app.js - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

SinglePageOne.html | AngularSeconds.html | FourThree.html | fiveTwo | FiveTwo.html | app.js

```
var myApp = angular.module("myApp", []);
// Register Service


myApp.service("RegisterService" , function(){
var uid = 1;
var users = [{
'id' : 0,
'name' : 'John Doe',
'email' : 'johndoe@gmail.com',
'password': 'johndoe',
'phone' : '123-45-678-901'}];

 // Save User
 this.save = function(user)
 {
 if(user.id == null)
 {
 user.id = uid++;
 users.push(user);
 }
 else
 {
 for(var i in users)
 {
 if(users[i].id == user.id)
 {
 users[i] = user;
 }
 }
 }
 };

 // Search User
 this.get = function(id)
 {
 for(var i in users )
 {
 if( users[i].id == id)
```

```
 // Search User
 this.get = function(id)
 {
 for(var i in users )
 {
 if( users[i].id == id)
 {
 return users[i];
 }
 }
 };

 // Delete User
 this.delete = function(id)
 {
 for(var i in users)
 {
 if(users[i].id == id)
 {
 users.splice(i,1);
 }
 }
 };

 // List Users
 this.list = function()
 {
 return users;
 };
});


Register Controller
myApp.controller("RegisterController" , function($scope , RegisterService){
console.clear();
$scope.ifSearchUser = false;
$scope.title ="User List";
$scope.users = RegisterService.list();
$scope.saveUser = function()
 {
```

```
$scope.saveUser = function()
{
 console.log($scope.newuser);
 if($scope.newuser == null || $scope.newuser == angular.undefined)
 return;
 RegisterService.save($scope.newuser);
 $scope.newuser = {};
};
$scope.delete = function(id)
{
 RegisterService.delete(id);
 if($scope.newuser != angular.undefined && $scope.newuser.id == id)
 {
 $scope.newuser = {};
 }
};
$scope.edit = function(id)
{
 $scope.newuser = angular.copy(RegisterService.get(id));
};
$scope.searchUser = function(){
 if($scope.title == "User List"){
 $scope.ifSearchUser=true;
 $scope.title = "Back";
 }
 else
 {
 $scope.ifSearchUser = false;
 $scope.title = "User List";
 }
};
});

var myApp = angular.module("myApp", []);
// Register Service
myApp.service("RegisterService" , function(){
var uid = 1;
```

```
var users = [{
'id' : 0,
'name' : 'John Doe',
'email' : 'johndoe@gmail.com',
'password': 'johndoe',
'phone' : '123-45-678-901'}];

 // Save User
 this.save = function(user)
 {
 if(user.id == null)
 {
 user.id = uid++;
 users.push(user);
 }
 else
 {
 for(var i in users)
 {
 if(users[i].id == user.id)
 {
 users[i] = user;
 }
 }
 }
 };

 // Search User
 this.get = function(id)
 {
 for(var i in users )
 {
 if( users[i].id == id)
 {
 return users[i];
 }
 }
 };
```

```
  // Delete User
  this.delete = function(id)
  {
  for(var i in users)
  {
  if(users[i].id == id)
  {
  users.splice(i,1);
  }
  }
  };

  // List Users
  this.list = function()
  {
  return users;
  };
 });

 // Register Controller
 myApp.controller("RegisterController" , function($scope , RegisterService){
 console.clear();
 $scope.ifSearchUser = false;
 $scope.title ="User List";
 $scope.users = RegisterService.list();
 $scope.saveUser = function()
 {
  console.log($scope.newuser);
  if($scope.newuser == null || $scope.newuser == angular.undefined)
  return;
  RegisterService.save($scope.newuser);
  $scope.newuser = {};
 };
 $scope.delete = function(id)
 {
  RegisterService.delete(id);
  if($scope.newuser != angular.undefined && $scope.newuser.id == id)
  {
  $scope.newuser = {};
```

```
RegisterService.delete(id);
if($scope.newuser != angular.undefined && $scope.newuser.id == id)
{
$scope.newuser = {};
}
;
scope.edit = function(id)

$scope.newuser = angular.copy(RegisterService.get(id));
;
scope.searchUser = function(){
if($scope.title == "User List"){
$scope.ifSearchUser=true;
$scope.title = "Back";
}
else
{
$scope.ifSearchUser = false;
$scope.title = "User List";
}
;
);
```

Output

User Registraion Form - W3Adda

m

{{title}}

Name:

John Doe

Email:

johndoe@gmail.com

Password:

•••••••

Phone:

123-45-678-901

Submit

Registered Users

Name	Email	Phone	Action
{{ user.name }}	{{ user.email }}	{{ user.phone }}	edit delete

No Users Found

EXPERIMENT 7

AIM: Create an application to demonstrate mouse and keyboard events directives.

OBJECTIVE: Generally while developing applications different type of html DOM events like mouse clicks, key press, change events, etc can be used likewise angularjs is having its own event directives for DOM interactions.In angularjs different type of DOM event listener directives are available and which can attach those events to html elements. THEORY: AngularJS includes certain directives which can be used to provide custom behavior on various DOM events, such as click, dblclick, mouseenter etc. The following table lists AngularJS event directives.

Event Directive

ng-blur

ng-change

ng-click

ng-dblclick

ng-focus

ng-keydown

ng-keyup

ng-keypress

ng-mousedown

ng-mouseenter

ng-mouseleave

ng-mousemove

ng-mouseover

ng-mouseup

ng-click : The ng-click directive is used to provide event handler for click event. ng-dblclick : The directive ng-dblclick in AngularJS invokes whenever an element with which ng-dblclick is attached is double-clicked. Angular JS will not override the element‟s original. ng-focus: This directive will execute the particular code when the user focuses on the element with which the ng-focus directive is attached. ng-blur : This directive will execute the particular code when a user loses focuses from the element with which ng-blur directive attached

mouse events : It occurs when the control of cursor moves over an element or an element is clicked by mouse event. The order of mouse event when the cursor moves over an element is:

ng-mouseover • ng-mouseenter • ng-mousemove • ng-mouseleave The order of mouse event when the mouse clicks on an element • ng-mousedown • ng-mouseup • ng-click $event Object: passed as an argument, when calling a function. The $event object contains the Browser‟s event.

```
<!DOCTYPE html>
<html>
<head>
 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"
></script>
<style>
 .redDiv {
 width: 100px;
 height: 100px;
 background-color: red;
 padding:2px 2px 2px 2px;
 }
 .yellowDiv {
 width: 100px;
 height: 100px;
 background-color: yellow;
 padding:2px 2px 2px 2px;
 }
 </style>
 </head>
<body ng-app="myApp" >
<h1>AngularJS ng-click Demo: </h1>
 <div ng-controller="myController">
 Enter Password: <input type="password" ng-model="password" />
<br /><br />
<button ng-click="DisplayMessage(password)">Show Password</button
</div>
96
 <h1>AngularJS Mouse Events Demo: </h1>
 <div ng-class="{redDiv: enter, yellowDiv: leave}" ng<0x02>mouseenter="enter=true;leave=false;" ng<0x02>mouseleave="leave=true;enter=false
 Mouse <span ng-show="enter">Enter</span> <span ng<0x02>show="leave">Leave</span>
 </div>
 <script>
 var myApp = angular.module('myApp', []);
 myApp.controller("myController", function ($scope, $window) {
 $scope.DisplayMessage = function (value) {
alert(value)
 }
 });
```

AngularJS ng-click Demo:

Enter Password: ••••••

Show Password 96

AngularJS Mouse Events Demo:

Mouse Leave

This page says

111111

OK

EXPERIMENT 8

Aim: Demonstrate controllers in Angular.js through an application a) Programming Controllers & $scope object

Objective: Create an application that needs to set up the initial state for the AngularJS $scope. set up the initial state of a scope by attaching properties to the $scope object.

THEORY: The controller in AngularJS is a JavaScript function that maintains the application data and behavior using $scope object. This can attach properties and methods to the $scope object inside a controller function, which in turn will add or update the data and attach behaviours to HTML elements. The $scope object is a glue between the controller and HTML. The ng-controller directive is used to specify a controller in an HTML element, which will add behavior or maintain the data in that HTML element and its child elements. The following example demonstrates attaching properties to the $scope object inside a controller and then displaying property value in HTML

F:\MumbaiUniversity\Nodejs\Practice\Seven\SevenOne.html (Seven) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
- Seven
  - AppData.js
  - SevenOne.html

SevenOne.html | AppData.js

```html
<!doctype html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Example - example-controller-spicy-2-production</title>
 <script src="//code.angularjs.org/snapshot/angular.min.js"></script>
 <script src="AppData.js"></script>
</head>
<body ng-app="spicyApp2">
 <div ng-controller="SpicyController">
<input ng-model="customSpice">
<button ng-click="spicy('chili')">Chili</button>
<button ng-click="spicy(customSpice)">Custom spice</button>
<p>The food is {{spice}} spicy!</p>
</div>
</body>
</html>
```

F:\MumbaiUniversity\Nodejs\Practice\Seven\AppData.js (Seven) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

Seven

AppData.js

SevenOne.html

SevenOne.html | AppData.js

```
(function(angular) {
 'use strict';
var myApp = angular.module('spicyApp2', []);
myApp.controller('SpicyController', ['$scope', function($scope) {
 $scope.customSpice = 'wasabi';
 $scope.spice = 'very';
$scope.spicy = function(spice) {
 $scope.spice = spice;
 };
}]);
})(window.angular);
```

Vert | Chili | Custom spice

The food is {{spice}} spicy!

EXPERIMENT No. 09

AIM: Demonstrate features of Simple Angular.js forms with a program

OBJECTIVE: Create Simple Angular Forms using different input controls & events.

THEORY: AngularJS facilitates you to create a form enriched with data binding and validation of input controls. Input controls are ways for a user to enter data. A form is a collection of controls for the purpose of grouping related controls together. Following are the input controls used in AngularJS forms: ● input elements ● select elements ● button elements ● textarea elements AngularJS provides multiple events that can be associated with the HTML controls. These events are associated with the different HTML input elements

Following is a list of events supported in AngularJS:

● ng-click

● ng-dbl-click

● ng-mousedown

● ng-mouseup

● ng-mouseenter

● ng-mouseleave

● ng-mousemove

● ng-mouseover

● ng-keydown

● ng-keyup

● ng-keypress

● ng-change

```
<!DOCTYPE html>
<html>
 <head>
 <title>Angular JS Forms</title>
 <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.j
s"></script>

 <style>
 table, th , td {
 border: 1px solid grey;
 border-collapse: collapse;
 padding: 5px;
 }

 table tr:nth-child(odd) {
 background-color: lightpink;
 }

 table tr:nth-child(even) {
 background-color: lightyellow;
 }
 </style>


 </head>
 <body>

 <h2>AngularJS Sample Application</h2>
 <div ng-app = "mainApp" ng-controller = "studentController">

 <form name = "studentForm" novalidate>
 <table border = "0">
 <tr>
 <td>Enter first name:</td>
 <td><input name = "firstname" type = "text" ng-model = "firstName" required>
 <span style = "color:red" ng-show = "studentForm.firstname.$dirty && studentForm.firstname.$invalid">
 <span ng-show = "studentForm.firstname.$error.required">First Name is required.</span>
 </span>
 </td>
```

```
 <tr>
 <td>Enter last name: </td>
 <td><input name = "lastname" type = "text" ng-model = "lastName" required>
 <span style = "color:red" ng-show = "studentForm.lastname.$dirty && studentForm.lastname.$invalid">
 <span ng-show = "studentForm.lastname.$error.required">Last Name is required.</span>
 </span>
 </td>
 </tr>

 <tr>
 <td>Email: </td><td><input name = "email" type = "email" ng-model = "email" length = "100" required>
 <span style = "color:red" ng-show = "studentForm.email.$dirty && studentForm.email.$invalid">
 <span ng-show = "studentForm.email.$error.required">Email is required.</span>
 <span ng-show = "studentForm.email.$error.email">Invalid email address.</span>
 </span>
 </td>
 </tr>
 <tr>

 <td>
 <button ng-click = "reset()">Reset</button>
 </td>
 <td>
 <button ng-disabled = "studentForm.firstname.$dirty && studentForm.firstname.$invalid || studentForm.lastname.$dirty &&
 studentForm.lastname.$invalid||studentForm.email.$dirty &&
 studentForm.email.$invalid" ng<0x02>click="submit()">Submit</button>
 </td>
 </tr>

 </table>
 </form>
 </div>
```

```
<script>
var mainApp = angular.module("mainApp", []);

mainApp.controller('studentController', function($scope) {
$scope.reset = function(){
$scope.firstName = "idol";
$scope.lastName = "mumbai university";
$scope.email = "mca@mu.ac.in";
}

$scope.reset();
});
</script>

</body>
/html>
```

AngularJS Sample Application

Enter first name:	idol
Enter last name:	mumbai university
Email:	mca@mu.ac.in
Reset	Submit

EXPERIMENT No. 10

AIM: Write a Angular.js program to implement the concept of Single page application.

OBJECTIVE: Create a single page application that loads a single HTML page and only a part of the page instead of the entire page gets updated with every click of the mouse.

THEORY: Single page applications or (SPAs) are web applications that load a single HTML page and dynamically update the page based on the user interaction with the web application

Single page application (SPA) is a web application that fits on a single page. All your code (JS, HTML, CSS) is retrieved with a single page load. And navigation between pages performed without refreshing the whole page. The page does not reload or transfer control to another page during the process. This ensures high performance and loading pages faster. Most modern applications use the concept of SPA. In the SPA, the whole data is sent to the client from the server at the beginning. As the client clicks certain parts on the webpage.This results in a lesser load on the server and is cost-efficient. SPAs use AJAX and HTML5 to create fluid and responsive Web applications and most of the work happens on the client-side. Popular applications such as Facebook, Gmail, Twitter, Google Drive, Netflix, and many more are examples of SPA.

Advantages:

1. Team collaboration: Single-page applications are excellent when more than one developer is working on the same project. It allows backend developers to focus on the API, while the frontend developers can focus on creating the user interface based on the backend API. 132

2. Caching: The application sends a single request to the server and stores all the received information in the cache. This proves beneficial when the client has poor network connectivity.

3. Fast and responsive: As only parts of the pages are loaded dynamically, it improves the website‟s speed.

- Debugging is easier Debugging single page applications with chrome is easier since such applications are developed using AngularJS Batarang and React developer tools.

- Linear user experience Browsing or navigating through the website is easy. Disadvantages:

1. SEO optimization: SPAs provide poor SEO optimization. This is because single-page applications operate on JavaScript and load data at once server. The URL does not change and different pages do not have a unique URL. Hence it is hard for the search engines to index the SPA website as opposed to traditional server-rendered pages.

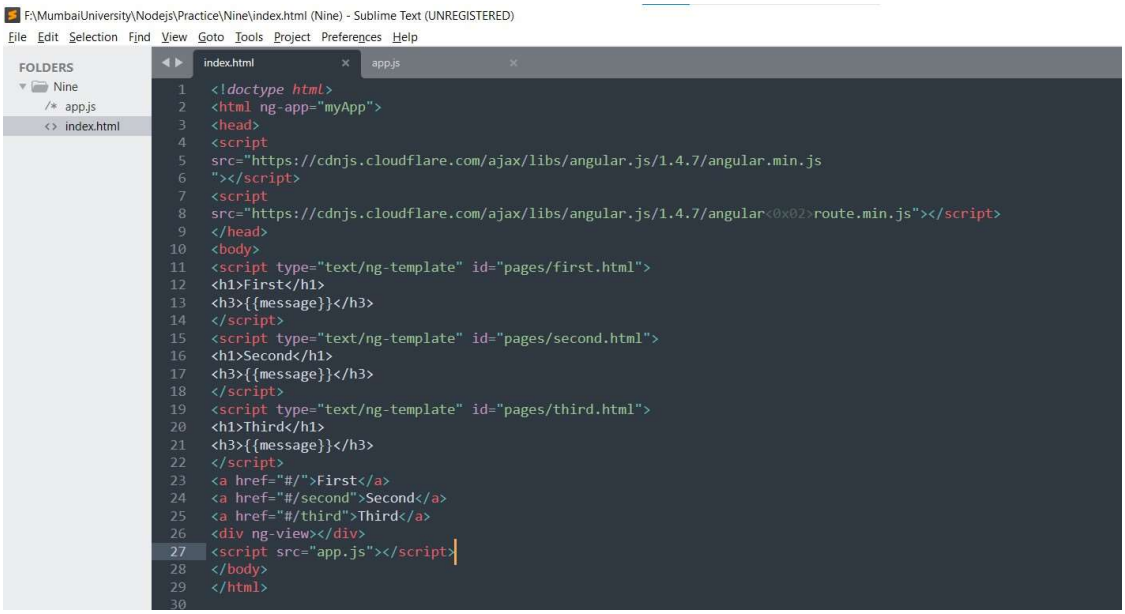
2. Browser history: A SPA does not save the users‟ transition of states within the website. A browser saves the previous pages only, not the state transition. Thus when users click the

back button, they are not redirected to the previous state of the website. To solve this problem, developers can equip their SPA frameworks with the HTML5 History API.

3. Security issues: Single-page apps are less immune to cross-site scripting (XSS) and since no new pages are loaded, hackers can easily gain access to the website and inject new scripts on the client-side.

4. Memory Consumption: Since the SPA can run for a long time, sometimes hours at a time, one needs to make sure the application does not consume more memory than it needs. Else, users with low memory devices may face serious performance issues.

5. Disabled Javascript: Developers need to chalk out ideas for users to access the information on the website for browsers that have Javascript disabled.



App.js

F:\MumbaiUniversity\Nodejs\Practice\Nine\app.js (Nine) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
- Nine
  - app.js
  - index.html

index.html | app.js

```
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: '/'});
        });

        <!-- controller is a JS function that maintains
    application data and behavior using $scope object -->
        <!--properties and methods can be attached to the
            $scope object inside a controller function-->

        app.controller('FirstController', function($scope) {
        $scope.message = 'Hello from FirstController';
        });

        app.controller('SecondController', function($scope) {
        $scope.message = 'Hello from SecondController';
        });

        app.controller('ThirdController', function($scope) {
        $scope.message = 'Hello from ThirdController';
```

First Second Third

First Page

Welcome to GeeksForGeeks

Hello from FirstController