JSS MAHAVIDYAPEETA

JSS ACADEMY OF TECHNICAL EDUCATION, BENGALURU

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

Laboratory Manual

Three Days Faculty Development Program

on

"Angular and NodeJS"

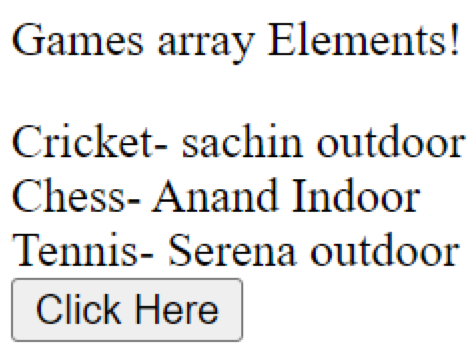
2nd – 4th November 2023

1. **Write an AngularJS Script to display list of games stored in an array on click of button using ng-click. Also demonstrate ng-init, ng-bing directives of AngularJS**

Execution Steps:
Step 1: Save the following program with .html extension
Step 2: Open the file using any browser

```
<html>
   <head>
          <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js">
          </script>
          <script>
                 var app = angular.module("myApp", []);
                 app.controller('userCtrl', ['$scope', function ($scope) {
                 $scope.showGames = function () {
                 $scope.games = [{ GameName: 'Cricket-', PlayerName:
'sachin', GameType: 'outdoor' },{ GameName: 'Chess-', PlayerName:
'Anand', GameType: 'Indoor' },{ GameName: 'Tennis-', PlayerName:
'Serena', GameType: 'outdoor' }];
                 }
                 }]);
          </script>
   </head>
   <body ng-app="myApp" ng-controller="userCtrl">
          <form>
                 <div ng-app="myApp" ng-init="ga='Games array
Elements!'">
                        <p ng-bind="ga"></p>
                 </div>
                 <div ng-controller="userCtrl">
                        <div ng-repeat='game in games'>
                               {{game.GameName}}
                               {{game.PlayerName}}
                               {{game.GameType}}
                        </div>
                 </div>
                 <input type="button" ng-click="showGames()" value="Click
Here"> </button>
          </form>
   </body>
</html>
```

Output:



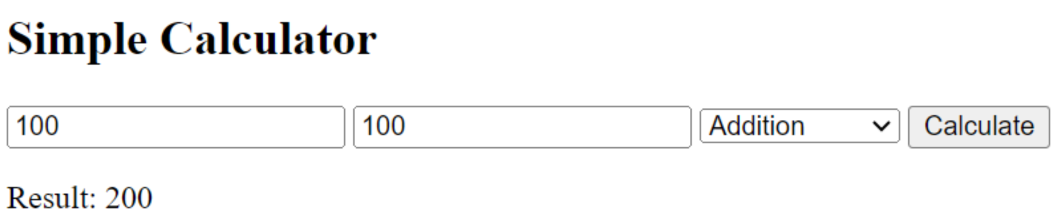
```
Games array Elements!

Cricket- sachin outdoor
Chess- Anand Indoor
Tennis- Serena outdoor
```

2. Write an AngularJS script to demonstrate simple calculator

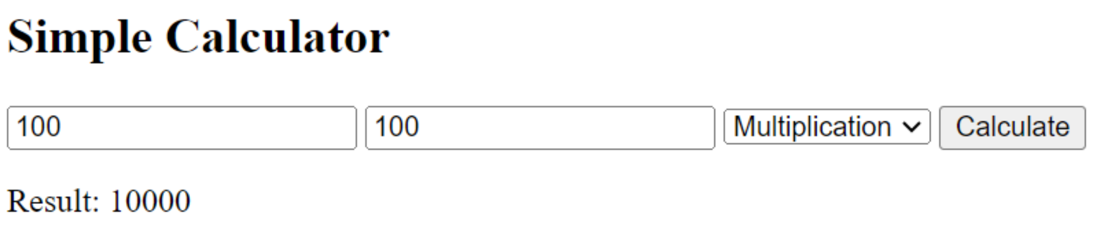
Execution Steps:
Step 1: Save the following program with .html extension
Step 2: Open the file using any browser

```
<!DOCTYPE html>
<html ng-app="calculatorApp">
<head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js"></script>
</head>
<body ng-controller="calculatorController">
  <h2>Simple Calculator</h2>
  <input type="number" ng-model="num1" placeholder="Enter number 1">
  <input type="number" ng-model="num2" placeholder="Enter number 2">
  <select ng-model="operator">
    <option value="+">Addition</option>
    <option value="-">Subtraction</option>
    <option value="*">Multiplication</option>
    <option value="/">Division</option>
  </select>
  <button ng-click="calculate()">Calculate</button>
  <p>Result: {{ result }}</p>

  <script>
    var app = angular.module('calculatorApp', []);
    app.controller('calculatorController', function($scope) {
      $scope.num1 = 0;
      $scope.num2 = 0;
      $scope.operator = '+';
      $scope.result = 0;

      $scope.calculate = function() {
        if ($scope.operator === '+') {
          $scope.result = $scope.num1 + $scope.num2;
        } else if ($scope.operator === '-') {
          $scope.result = $scope.num1 - $scope.num2;
        } else if ($scope.operator === '*') {
          $scope.result = $scope.num1 * $scope.num2;
        } else if ($scope.operator === '/') {
          if ($scope.num2 === 0) {
            $scope.result = 'Cannot divide by zero';
          } else {
            $scope.result = $scope.num1 / $scope.num2;
          }
        }
      };
    });
  </script>
</body>
</html>
```

Output:



Result: 200



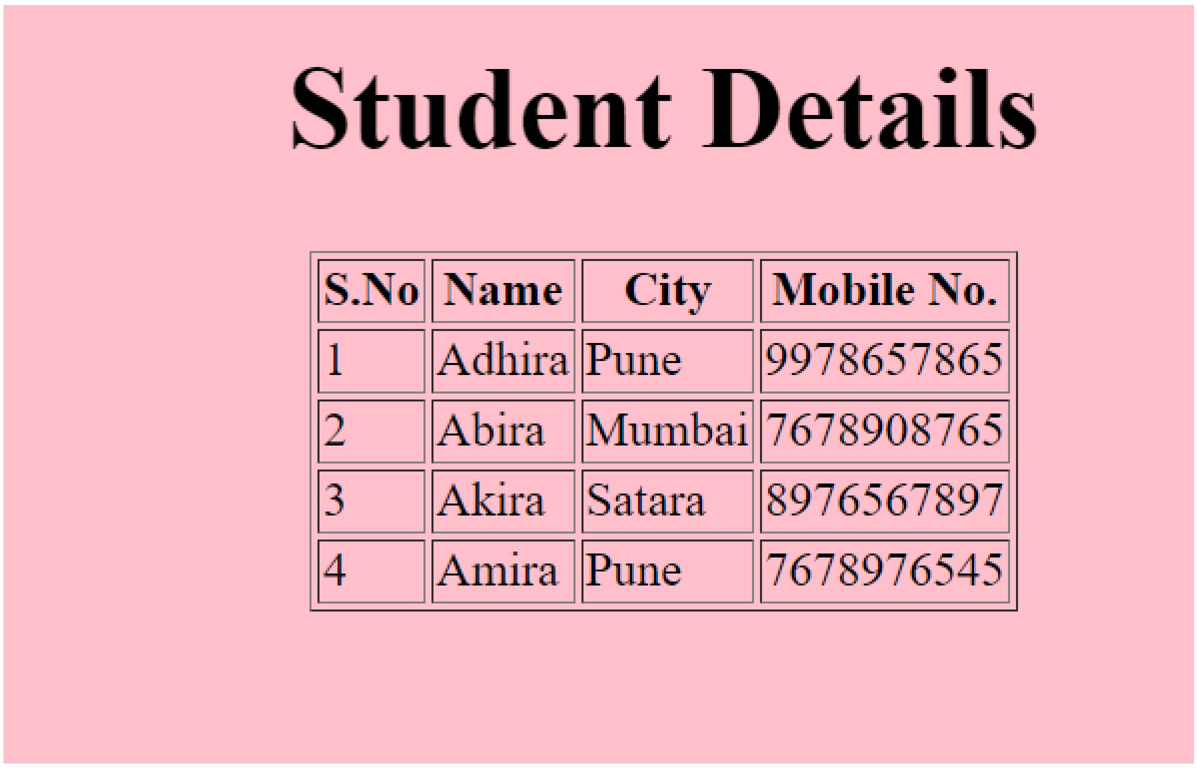
Result: 10000

3. Using AngularJS display the 10 students details in table format.

Execution Steps:
Step 1: Save the following program with .html extension
Step 2: Open the file using any browser

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js"></script>
<!-- ng-repeat- it used to repeat a set of HTML code for a number of times-
->
<style>
body {
   margin: 2%;
   font-size: 120%;
   background-color: pink;
}
</style>

<body ng-app="myApp" ng-controller="ListController">
   <h1 align='center'>Student Details</h1>
   <table border=1 align='center'>
      <thead>
         <tr>
            <th>S.No</th>
            <th>Name</th>
            <th>City</th>
            <th>Mobile No.</th>
         </tr>
      </thead>
      <tr ng-repeat="item in itemsDetails">
         <td>{{item.sno}}</td>
         <td>{{item.name}}</td>
         <td>{{item.city}}</td>
         <td>{{item.mobileno}}</td>
      </tr>
   </table>
</body>

<script>
   var app = angular.module('myApp', []);
   app.controller('ListController', function($scope) {
      $scope.itemsDetails = [{
            sno: 1,
            name: 'Adhira',
            city: 'Pune',
            mobileno: 9978657865,
            },
            {
                sno: 2,
                name: 'Abira',
                city: 'Mumbai',
                mobileno: 7678908765,
            },
            {
                sno: 3,
                name: 'Akira',
                city: 'Satara',
                mobileno: 8976567897,
            },
            {
                sno: 4,
                name: 'Amira',
                city: 'Pune',
                mobileno: 7678976545,
            }
        ];
    });
</script>
</html>
```

Output:



Student Details

S.No	Name	City	Mobile No.
1	Adhira	Pune	9978657865
2	Abira	Mumbai	7678908765
3	Akira	Satara	8976567897
4	Amira	Pune	7678976545

4. Create an HTML form using AngularJS that contains the student registration details and validate student first and last name as it should not contain other than alphabets and age should be between 18 to 50 and display greeting message depending on current time using ng-show.

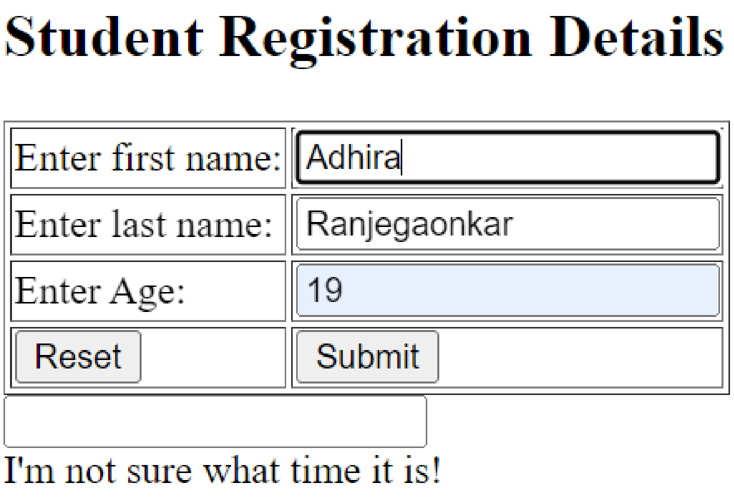
Execution Steps:
Step 1: Save the following program with .html extension
Step 2: Open the file using any browser

```
<!DOCTYPE html>
<html ng-app="mainApp">
<head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js"></script>
</head>
<body ng-controller="studentController">
  <h2>Student Registration Details</h2>
  <div>
    <form name="studentForm" novalidate>
      <table border="1">
        <tr>
          <td>Enter first name:</td>
          <td>
            <input name="firstName" type="text" ng-model="firstName" ng-
pattern="/^[a-zA-Z]*$/" required>
            <span style="color:red" ng-show="studentForm.firstName.$dirty
&& studentForm.firstName.$invalid">
              <span ng-show="studentForm.firstName.$error.required">First
Name is required.</span>
              <span style="color:red" ng-
show="studentForm.firstName.$error.pattern">*Invalid First
name.</span>
            </span>
          </td>
        </tr>
        <tr>
          <td>Enter last name: </td>
          <td>
            <input name="lastName" type="text" ng-model="lastName" ng-
pattern="/^[a-zA-Z]*$/" 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 style="color:red" ng-show="studentForm.lastName.$error.pattern">*Invalid Last name.</span>
         </span>
        </td>
       </tr>
       <tr>
        <td>Enter Age:</td>
        <td>
         <input type="text" ng-model="age" name="age" ng-pattern="/^(?:1[8-9]|[2-5][0-9]|50)$/" required/>
         <span style="color:red" ng-show="studentForm.age.$dirty && studentForm.age.$invalid">
          <span ng-show="studentForm.age.$error.required">Age is required.</span>
          <span style="color:red" ng-show="studentForm.age.$error.pattern">*Invalid Age. Valid 18-50</span>
         </span>
        </td>
       </tr>
       <tr>
        <td>
         <button ng-click="reset()">Reset</button>
        </td>
        <td>
         <button ng-disabled="studentForm.$invalid" ng-click="submit()">Submit</button>
        </td>
       </tr>
      </table>
    </form>
  </div>
  <div>
    <input type="number" ng-model="value"><br>
    <span>{{ value | greet }}</span>
  </div>
  <script>
    var mainApp = angular.module("mainApp", []);
    mainApp.controller('studentController', function ($scope) {
      $scope.reset = function () {
        $scope.firstName = 'Adhira';
        $scope.lastName = 'Ranjegaonkar';
        $scope.age = '';
      }
      $scope.reset();
    });
```

```
        mainApp.filter('greet', function() {
          return function(input) {
            if (input < 12) {
              return 'Good Morning';
            } else if (input >= 12 && input <= 17) {
              return 'Good Afternoon';
            } else if (input > 17 && input <= 24) {
              return 'Good Evening';
            } else {
              return "I'm not sure what time it is!";
            }
          };
        });
      </script>
    </body>
    </html>
```

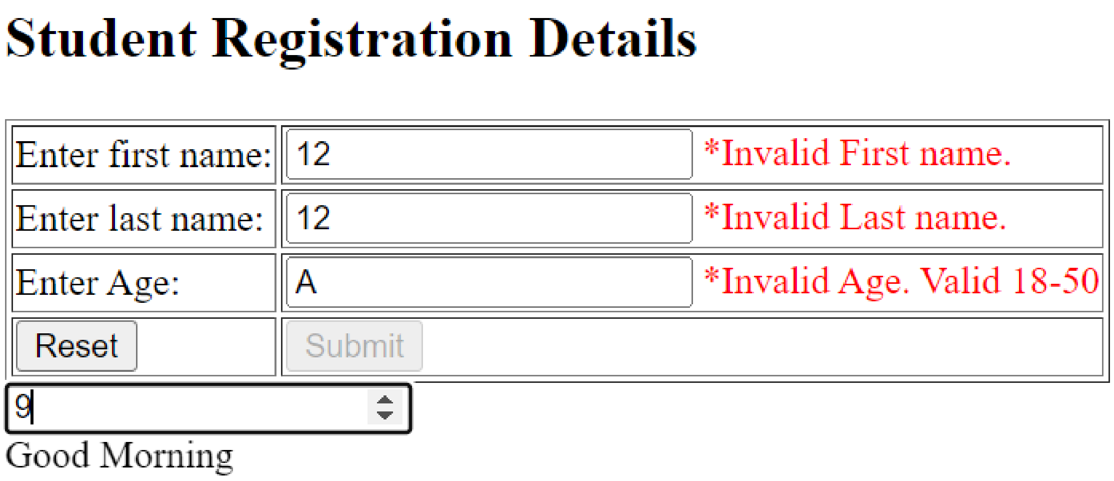
Output:



Student Registration Details

Enter first name:	Adhira
Enter last name:	Ranjegaonkar
Enter Age:	19
Reset	Submit

I'm not sure what time it is!



Student Registration Details

Enter first name:	12	*Invalid First name.
Enter last name:	12	*Invalid Last name.
Enter Age:	A	*Invalid Age. Valid 18-50
Reset	Submit	

9

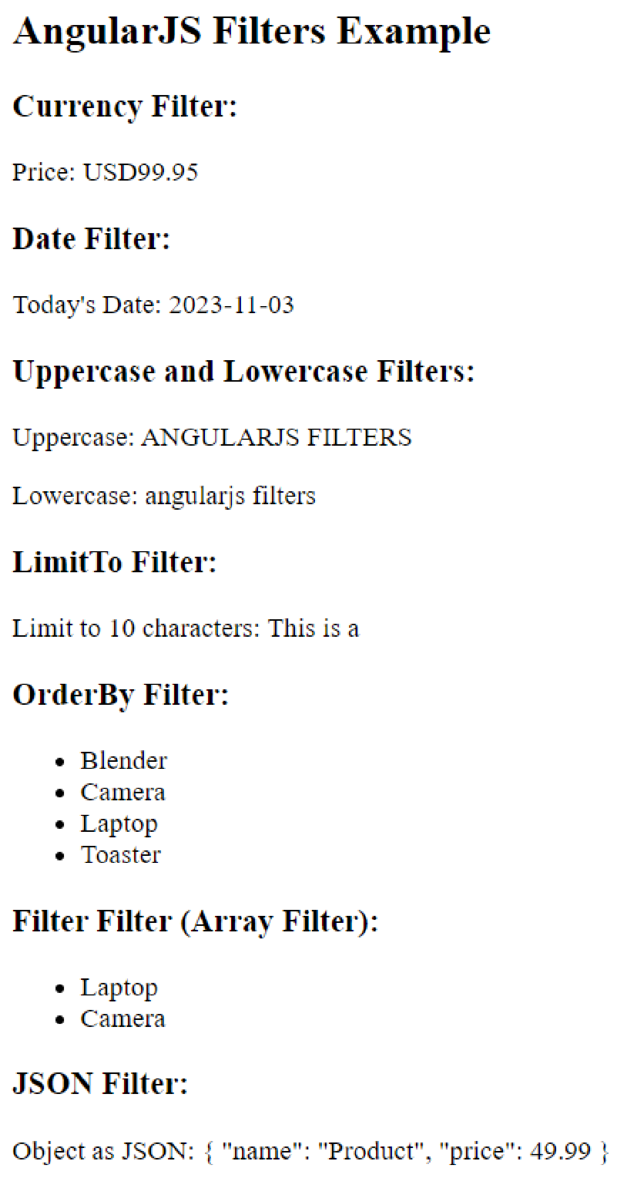
Good Morning

5. Write a program that demonstrates the use of AngularJS filters in an HTML.

Execution Steps:
Step 1: Save the following program with .html extension
Step 2: Open the file using any browser

```
<!DOCTYPE html>
<html ng-app="filterApp">
<head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js"></script>
</head>
<body ng-controller="filterController">
  <h2>AngularJS Filters Example</h2>

  <h3>Currency Filter:</h3>
  <p>Price: {{ price | currency: 'USD':2 }}</p>

  <h3>Date Filter:</h3>
  <p>Today's Date: {{ currentDate | date: 'yyyy-MM-dd' }}</p>

  <h3>Uppercase and Lowercase Filters:</h3>
  <p>Uppercase: {{ text | uppercase }}</p>
  <p>Lowercase: {{ text | lowercase }}</p>

  <h3>LimitTo Filter:</h3>
  <p>Limit to 10 characters: {{ longText | limitTo: 10 }}</p>

  <h3>OrderBy Filter:</h3>
  <ul>
    <li ng-repeat="item in items | orderBy: 'name'">{{ item.name }}</li>
  </ul>

  <h3>Filter Filter (Array Filter):</h3>
  <ul>
    <li ng-repeat="item in items | filter: { category: 'Electronics' }">{{
item.name }}</li>
  </ul>

  <h3>JSON Filter:</h3>
  <p>Object as JSON: {{ object | json }}</p>

  <script>
    var app = angular.module('filterApp', []);
    app.controller('filterController', function($scope) {
        $scope.price = 99.95;
        $scope.currentDate = new Date();
        $scope.text = 'AngularJS Filters';
        $scope.longText = 'This is a long text that should be limited.';
        $scope.items = [
          { name: 'Laptop', category: 'Electronics' },
          { name: 'Toaster', category: 'Appliances' },
          { name: 'Camera', category: 'Electronics' },
          { name: 'Blender', category: 'Appliances' }
        ];
        $scope.object = { name: 'Product', price: 49.99 };
      });
    </script>
  </body>
</html>
```

Output:



AngularJS Filters Example

Currency Filter:

Price: USD99.95

Date Filter:

Today's Date: 2023-11-03

Uppercase and Lowercase Filters:

Uppercase: ANGULARJS FILTERS

Lowercase: angularjs filters

LimitTo Filter:

Limit to 10 characters: This is a

OrderBy Filter:

- Blender
- Camera
- Laptop
- Toaster

Filter Filter (Array Filter):

- Laptop
- Camera

JSON Filter:

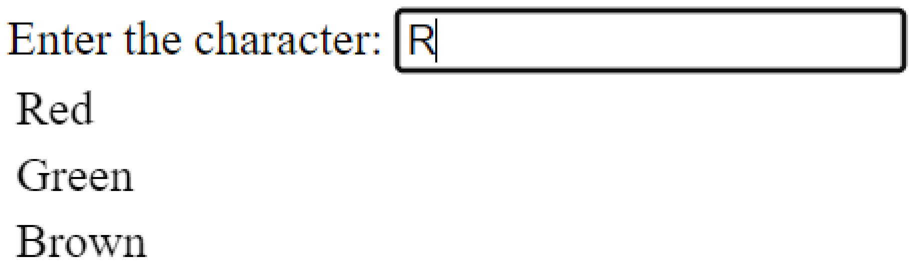
Object as JSON: { "name": "Product", "price": 49.99 }

6. Write an AngularJS script to search color name according to the character typed.

Execution Steps:
Step 1: Save the following program with .html extension
Step 2: Open the file using any browser

```
<!DOCTYPE html>
<html ng-app="mainApp">
<head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js"></script>
</head>
<body ng-controller="FilterController">
  <div>
    Enter the character:
    <input type="text" name="searchname" ng-model="searchname">
  </div>

  <div ng-hide="!searchname">
    <table>
      <tr ng-repeat="a in users | filter: searchname">
        <td>{{ a.name }}</td>
      </tr>
    </table>
  </div>

  <script>
    var mainApp = angular.module("mainApp", []);
    mainApp.controller("FilterController", function($scope) {
      $scope.users = [{
        name: "Red",
      }, {
        name: "Yellow",
      }, {
        name: "Green",
      }, {
        name: "Brown",
      }, {
        name: "Pink",
      }];
    });
  </script>
</body>
</html>
```

Output:



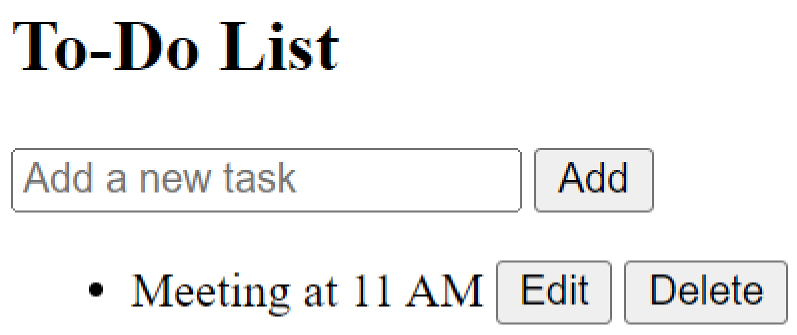
```
Enter the character: R
Red
Green
Brown
```

7. Create simple To-Do list application using AngularJS that allows users to add, edit and remove tasks.

Execution Steps:
Step 1: Save the following program with .html extension
Step 2: Open the file using any browser

```
<!DOCTYPE html>
<html ng-app="ToDoApp">
<head>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js"></script>
</head>
<body ng-controller="ToDoController">
  <h2>To-Do List</h2>

  <div>
    <input type="text" ng-model="newTask" placeholder="Add a new task">
    <button ng-click="addTask()">Add</button>
  </div>

  <ul>
    <li ng-repeat="task in tasks">
      {{ task.name }}
      <span ng-show="task.editing">
        <input type="text" ng-model="task.name" ng-blur="saveTask(task)">
      </span>
      <span ng-show="!task.editing">
        <button ng-click="editTask(task)">Edit</button>
        <button ng-click="removeTask(task)">Delete</button>
      </span>
    </li>
  </ul>

  <script>
    var app = angular.module('ToDoApp', []);

    app.controller('ToDoController', function($scope) {
      $scope.tasks = [];

      $scope.addTask = function() {
        if ($scope.newTask) {
          $scope.tasks.push({ name: $scope.newTask, editing: false });
          $scope.newTask = '';
        }
      };
      $scope.editTask = function(task) {
        task.editing = true;
      };

      $scope.saveTask = function(task) {
        task.editing = false;
      };

      $scope.removeTask = function(task) {
        const index = $scope.tasks.indexOf(task);
        if (index !== -1) {
          $scope.tasks.splice(index, 1);
        }
      };
    });
  </script>
</body>
</html>
```

Output:



To-Do List

8. Create a Server using NodeJs.

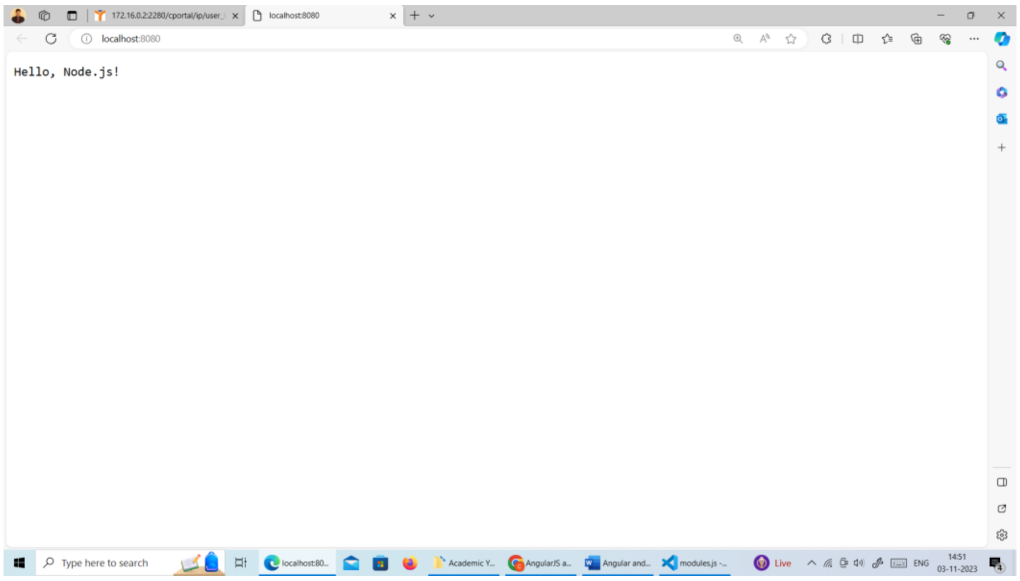
Execution Steps:

Step 1: Create a new file with **.js** extension

Step 2: run the file in the terminal using the command **node filename.js**

```
var http=require('http');
var server = http.createServer(function(req,res){
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, Node.js!');
});
server.listen(8080, 'localhost', () => {
    console.log('Server is running at http://localhost:8080/');
  });
```

Output:
Server is running at http://localhost:8080/



```
Hello, Node.js!
```

9. Create Clusters using NodeJS.

Simple Cluster

Execution Steps:

Step 1: Create a new file with **.js** extension

Step 2: run the file in the terminal using the command **node filename.js**

```
var cluster = require('cluster');
if (cluster.isWorker){
    console.log('Child Thread')

}else {
    console.log('Parent thread');
    cluster.fork();
    cluster.fork();
}
```

Output:

Parent thread

Child Thread

Child Thread

Cluster Performing User Specific Task

```
const cluster = require('cluster');
const http = require('http');
const fs = require('fs');

if (cluster.isWorker) {
    // Determine the worker's ID
    const workerId = cluster.worker.id;

    if (workerId === 1) {
        console.log(`Worker ${workerId} is creating a server`);
        // Create an HTTP server
        http.createServer((req, res) => {
            res.writeHead(200, { 'Content-Type': 'text/plain' });
            res.end('Hello, this is the server!\n');
        }).listen(8000);
    } else if (workerId === 2) {
        console.log(`Worker ${workerId} is creating a file`);
        // Create a file and write some data to it
        fs.writeFile('example.txt', 'This is some data for the file.', (err)
=> {
```

```
            if (err) {
                console.error('Error writing to file:', err);
            }
        });
    } else {
        console.log(`Worker ${workerId} is performing a specific task`);
        // Your code for a specific task goes here
    }
} else {
    console.log('Parent thread');

    // Create multiple workers
    for (let i = 1; i <= 3; i++) {
        cluster.fork();
    }
}
```

Output:

```
Parent thread

Worker 1 is creating a server

Worker 2 is creating a file

Worker 3 is performing a specific task
```

10. Program for creating the files and read contents from the file.

Execution Steps:

Step 1: Create a new file with **.js** extension

Step 2: run the file in the terminal using the command **node filename.js**

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

// Create a file and write data to it
fs.writeFile('example.txt', 'Hello, this is some text!', (err) => {
  if (err) {
    console.error('Error writing to file:', err);
  } else {
    console.log('File created and data written successfully.');

    // Read data from the file
    fs.readFile('example.txt', 'utf8', (err, data) => {
      if (err) {
        console.error('Error reading from file:', err);
      } else {
        console.log('Data read from the file:', data);
      }
    });
  }
});
```

Output:

```
File created and data written successfully.

Data read from the file: Hello, this is some text!
```

11. Uploading a file to a Node.js server

Execution Steps:

Step 1: Create a new directory for your project.

Step 2: Navigate to the directory and execute the command **npm init -y**

Step 3: Install the project dependencies using the command

npm install express multer

Step 4: Create nodejs server with file name **app.js**

```
const express = require('express');
const multer = require('multer');
const app = express();
const port = 3000;

// Set up a storage engine for multer
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    // Define the destination directory for uploaded files
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    // Define the filename for uploaded files
    cb(null, Date.now() + '-' + file.originalname);
  }
});
const upload = multer({ storage: storage });
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html'); // Create this HTML file in the same
directory
});
app.post('/upload', upload.single('file'), (req, res) => {
  res.send('File uploaded successfully!');
});
```

```
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Step 6: Create a HTML file with filename index.html

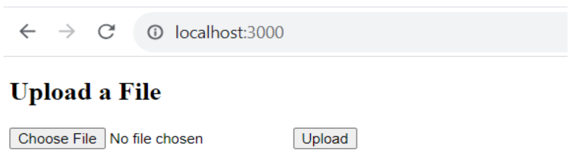
```
<!DOCTYPE html>
<html>
<body>
  <h2>Upload a File</h2>
  <form action="/upload" method="POST" enctype="multipart/form-data">
    <input type="file" name="file">
    <input type="submit" value="Upload">
  </form>
</body>
</html>
```

Step 7: Create a directory for storing the file uploads using command **mkdir foldername**

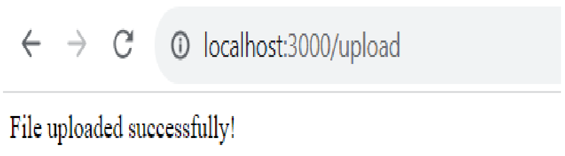
Step 8: Run the program using command **node app.js**

Step 9: Open the web browser and in URL mention http://localhost:3000 which will ask for uploading the file

Output:


localhost:3000

Upload a File


localhost:3000/upload

File uploaded successfully!

12. Program for connecting to MySql Database using NodeJS.

Execution Steps:

Step 1: Create a mysql database in MySql Server.

Step 2: Install the project dependencies using the command

npm install mysql

Step 3: create a node js script with file name **filename,js**

Step 4: Run the program using command **node filename.js**

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

var con = mysql.createConnection({
  host: "127.0.0.1",
  user: "root",
  password: "root",
  database: "nodejsexample",
  port:3306
});

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:

```
Connected!

Table created
```

13. Program to create a form and connect to database using the nodejs and express js using MVC.

Step 1: Initialize Your Node.js Project:

Create a new directory for your project, navigate to it in your terminal, and run the following command to initialize a new Node.js project with a package.json file:

npm init -y

Step 2: Install Required Dependencies:

npm install express mysql body-parser ejs

Step 3: Create a Directory Structure:

Organize your project files into directories. For example, you can create directories like "views," "public" for static files (e.g., CSS and JavaScript), and "models" for database models.

Step 4: Set Up Express.js:

Create an Express.js application in your main JavaScript file (e.g., app.js or server.js).

```
const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const app = express();


app.set('view engine', 'ejs');

app.use(bodyParser.urlencoded({ extended: true }));

app.use(express.static('public'));


// Connect to your MongoDB database

mongoose.connect('mongodb://localhost/your-database-name', {
useNewUrlParser: true });


// Create a schema and a model for the registration data

const registrationSchema = new mongoose.Schema({

  username: String,
  email: String,
  // Add more fields as needed
});

const Registration = mongoose.model('Registration', registrationSchema);

// Set up your routes here

// Start your server
app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Step 5: Create a Registration Form View:

Create an ejs view file in your "views" directory for the registration form.

```
<!-- views/registration.ejs -->
<!DOCTYPE html>
<html>
<head>
  <title>Registration Form</title>
</head>
<body>
  <h1>Registration Form</h1>
  <form action="/register" method="POST">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required><br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br>
    <button type="submit">Register</button>
  </form>
</body>
</html>
```

Step 6: Set Up Routes:

In your Express.js application, set up routes to handle GET and POST requests for the registration form. You will validate and save the registration data to the Mysql database.

```
// Define a route to render the registration form
app.get('/registration', (req, res) => {
  res.render('registration');
});


// Define a route to handle form submissions
app.post('/register', (req, res) => {
  const { username, email } = req.body;


  // Insert registration data into the MySQL database
  const insertSQL = 'INSERT INTO registrations (username, email) VALUES
(?, ?)';
  db.query(insertSQL, [username, email], (err) => {
    if (err) {
      console.error('Error inserting data into the database:', err);
    } else {
      console.log('Registration data saved to the database.');
      res.redirect('/registration');
    }
  });
});
```

Step 7: Start Your Server:

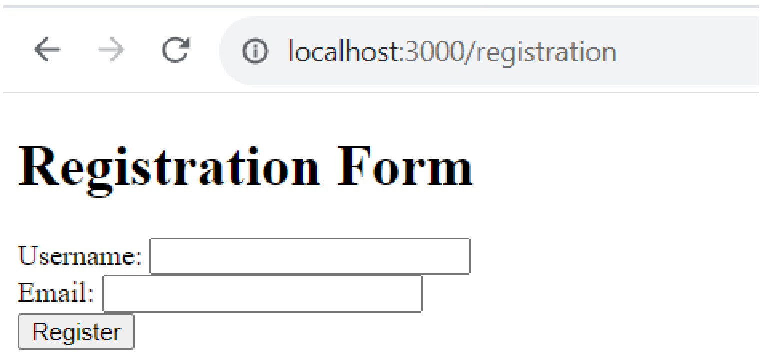
Run your Node.js application:

```
node app.js
```

Access the registration form at **http://localhost:3000/registration**.

Output:

```
Server is running on port 3000
Connected to MySQL database
```



Registration Form

Username:

Email:

Registration data saved to the database.