



SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)
SRM Nagar, Kattankulathur-603203.



DEPARTMENT OF COMPUTER APPLICATIONS

LAB MANUAL

MC4266 - FULL STACK WEB DEVELOPMENT LABORATORY

ACADEMIC YEAR 2024-2025

(EVEN SEMESTER)

SECOND SEMESTER

Prepared By

Mr. M. Asan Nainar, Asst.Prof. / MCA
Mr. N. Leo Bright Tennisson, Asst.Prof. / MCA,

INDEX

E.No.	Experiment Name	Page. No.
a	Syllabus	3
b	Lab Equipments	4
c	PE0,PO,PSO, CO, CO-PO Mapping	5
d	Mode of Assessment	7
1	Create a form and validate the contents of the form using JavaScript.	8
2	Get data using Fetch API from an open-source endpoint and display the contents in the form of a card.	12
3	Create a NodeJS server that serves static HTML and CSS files to the user without using Express.	15
4	Create a NodeJS server using Express that stores data from a form as a JSON file and displays it in another page. The redirect page should be prepared using Handlebars.	17
5	Create a NodeJS server using Express that creates, reads, updates and deletes students' details and stores them in MongoDB database. The information about the user should be obtained from a HTML form.	19
6	Create a NodeJS server that creates, reads, updates and deletes event details and stores them in a MySQL database. The information about the user should be obtained from a HTML form.	21
7	Create a counter using ReactJS	23
8	Create a Todo application using ReactJS. Store the data to a JSON file using a simple NodeJS server and retrieve the information from the same during page reloads.	25
9	Create a simple Sign up and Login mechanism and authenticate the user using cookies. The user information can be stored in either MongoDB or MySQL and the server should be built using NodeJS and Express Framework.	29
10	Create and deploy a virtual machine using a virtual box that can be accessed from the host computer using SSH.	33
11	Create a docker container that will deploy a NodeJS ping server using the NodeJS image.	35
Lab Viva Questions		38
Topic Beyond Syllabus		
Servlet application to display the examination result of a student		41

MC4266 FULL STACK WEB DEVELOPMENT LABORATORY**L T P C****0 0 4 2****COURSE OBJECTIVES:**

- To implement the client side of the web application using javascript.
- To understand Javascript on the desktop using NodeJS.
- To develop a web application using NodeJS and Express.
- To implement a SPA using React.
- To develop a full stack single page application using React, NodeJS, and aDatabase (MongoDB or SQL).

LIST OF EXPERIMENTS:

1. Create a form and validate the contents of the form using JavaScript.
2. Get data using Fetch API from an open-source endpoint and display the contents in the form of a card.
3. Create a NodeJS server that serves static HTML and CSS files to the user without usingExpress.
4. Create a NodeJS server using Express that stores data from a form as a JSON file and displays it in another page. The redirect page should be prepared using Handlebars.
5. Create a NodeJS server using Express that creates, reads, updates and deletes students' details and stores them in MongoDB database. The information about the usershould be obtained from a HTML form.
6. Create a NodeJS server that creates, reads, updates and deletes event details and storesthem in a MySQL database. The information about the user should be obtained from a HTML form.
7. Create a counter using ReactJS
8. Create a Todo application using ReactJS. Store the data to a JSON file using a simple NodeJS server and retrieve the information from the same during page reloads.
9. Create a simple Sign up and Login mechanism and authenticate the user using cookies.The user information can be stored in either MongoDB or MySQL and the server shouldbe built using NodeJS and Express Framework.
10. Create and deploy a virtual machine using a virtual box that can be accessed from the host computer using SSH.
11. Create a docker container that will deploy a NodeJS ping server using the NodeJS image.

TOTAL: 60 PERIODS

LAB EQUIPMENT FOR A BATCH OF 30 STUDENTS:**SOFTWARE REQUIREMENTS**

1. NodeJS/Express JS, ReactJS, Docker, any IDE like NOTEPAD++ / visual studio code/sublime text etc.,
2. MySQL, MongoDB

COURSE OUTCOMES:

CO1: To implement and deploy the client side of the web application.

CO2: To develop and deploy server side applications using NodeJS.

CO3: To use Express framework in web development.

CO4: To implement and architect database systems in both NoSQL and SQL environments.

CO5: To develop a full stack single page application using React, NodeJS, and a Database and deploy using containers.

PROGRAMME EDUCATIONAL COURSE OBJECTIVES (PEOs):

1. To prepare students with a breadth of knowledge to comprehend, analyze, design, and create computing solutions to real-life problems and to excel in industry / technical profession.
2. To provide students with a solid foundation in mathematical and computing fundamentals and techniques required to solve technology-related problems and to pursue higher studies and research.
3. To inculcate a professional and ethical attitude in students, to enable them to work towards a broad social context.
4. To empower students with skills required to work as members and leaders in multidisciplinary teams and with continuous learning ability on technology and trends needed for a successful career.

PROGRAM OUTCOMES (POs) MASTER'S IN COMPUTER APPLICATIONS GRADUATES WILL BE ABLE TO:

PO#	PROGRAMME COURSE OUTCOMES
1.	An ability to independently carry out research/investigation and development work to solve practical problems.
2.	An ability to write and present a substantial technical report/document.
3.	An ability to demonstrate a degree of mastery over the design and development of computer applications.
4.	An ability to create, select, adapt and apply appropriate innovative techniques, resources, and modern computing tools to complex computing activities with an understanding of the limitations.
5.	An ability to recognize the need and to engage in independent learning for continual development as a computing professional.
6.	An ability to function effectively as an individual and as a member/leader of a team in various technical environments.

PROGRAM SPECIFIC OUTCOMES (PSOs):

- 1) **Database Administrator:** Responsible for planning, design and development of web applications including installing, configuring, upgrading, monitoring, maintaining and security of web applications in an organization.
- 2) **Developer/Programmer:** Design and develop software solutions for modern business environments with the help of advanced computing technologies and programming tools.
- 3) **Network Administrator:** Design, plan and setting up the network that is helpful in contemporary business environments.
- 4) **Software Engineer/Tester:** Planning, defining test activities and preparing test cases that can identify errors using predefined procedures and ensure that the product maintains quality.

COURSE OUTCOMES:**Course Name: C4266&FSWDL****Lab Year of Study:2024-2025**

MC4266.1	Implement the client side of the web application.
MC4266.2	Deploy server side applications using NodeJS.
MC4266.3	Use Express framework in web development.
MC4266.4	Implement and architect database systems.
MC4266.5	Develop a full stack single page application.

CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	3	1	3	2	3	3
2	2	1	2	2	3	3
3	2	1	2	2	3	2
4	2	1	3	2	2	3
5	2	1	2	2	2	2
Avg	2.2	1	2.4	2	2.6	2.6

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Execution	30
2.	Record	10
3	Model Exam	20
Total		60

Ex.No.1**FORM VALIDATION USING JAVASCRIPT**

AIM: To create Form validation using JavaScript.

Procedure:**1. Project Setup**

- Create HTML Structure:
 - Create an HTML file (e.g., form_validation.html) with the following:
 - A form element (<form>) with an appropriate id (e.g., myForm).
 - Input fields for each data point (e.g., name, email, phone).
 - A submit button (<button type="submit">).

2. JavaScript Implementation

- Create a JavaScript file (e.g., script.js) and link it to your HTML file.
- Define Validation Functions:
 - Create separate functions for each validation rule (e.g., validateName(), validateEmail(), validatePhone()).
 - Implement the logic for each validation rule within the respective functions.
 - Example:
 - validateName(): Check if the name field is not empty and contains only letters and spaces.
 - validateEmail(): Check if the email field follows the standard email format (e.g., using a regular expression).
 - validatePhone(): Check if the phone number field contains only digits and is of the correct length.
- Event Handling:
 - Attach an event listener to the form's submit event.
 - Inside the event handler:
 - Call all the validation functions for each input field.
 - If any validation fails:
 - Display an appropriate error message (e.g., within the span element).
 - Prevent the form from submitting (using event.preventDefault()).
 - If all validations pass:
 - Submit the form (allow the default form submission behavior).

3. Test the Application**Program**

```
<html>
<head>
<script>
function VALIDATEDetail()
{
    var name = document.forms["RegForm"]["Name"];
    var email = document.forms["RegForm"]["EMail"];
    var phone = document.forms["RegForm"]["Telephone"];
    var what = document.forms["RegForm"]["Subject"];
    var password = document.forms["RegForm"]["Password"];
    var address = document.forms["RegForm"]["Address"];
```



```

if (name.value == "")
{
    window.alert("Please enter your name.");
    name.focus();
    return false;
}

if (address.value == "")
{
    window.alert("Please enter your address.");
    name.focus();
    return false;
}

if (email.value == "")
{
    window.alert("Please enter a valid e-mail address.");
    email.focus();
    return false;
}

if (email.value.indexOf("@", 0) < 0)
{
    window.alert("Please enter a valid e-mail address.");
    email.focus();
    return false;
}

if (email.value.indexOf(".", 0) < 0)
{
    window.alert("Please enter a valid e-mail address.");
    email.focus();
    return false;
}

if (phone.value == "")
{
    window.alert("Please enter your telephone number.");
    phone.focus();
    return false;
}

if (password.value == "")
{
    window.alert("Please enter your password");
    password.focus();
    return flase;
}

if (what.selectedIndex < 1)
{
    alert("Please enter your course.");
    what.focus();
    return false;
}

```

```

    }

    return true;
}</script>

<style>
VALIDATEDDETAIL {
    font-weight: bold ;
    float: left;
    width: 100px;
    text-align: left;
    margin-right: 10px;
    font-size:14px;
}

div {
box-sizing: border-box;
    width: 100%;
    border: 100px solid black;
    float: left;
    align-content: center;
    align-items: center;
}

form {
    margin: 0 auto;
    width: 600px;
}</style></head>

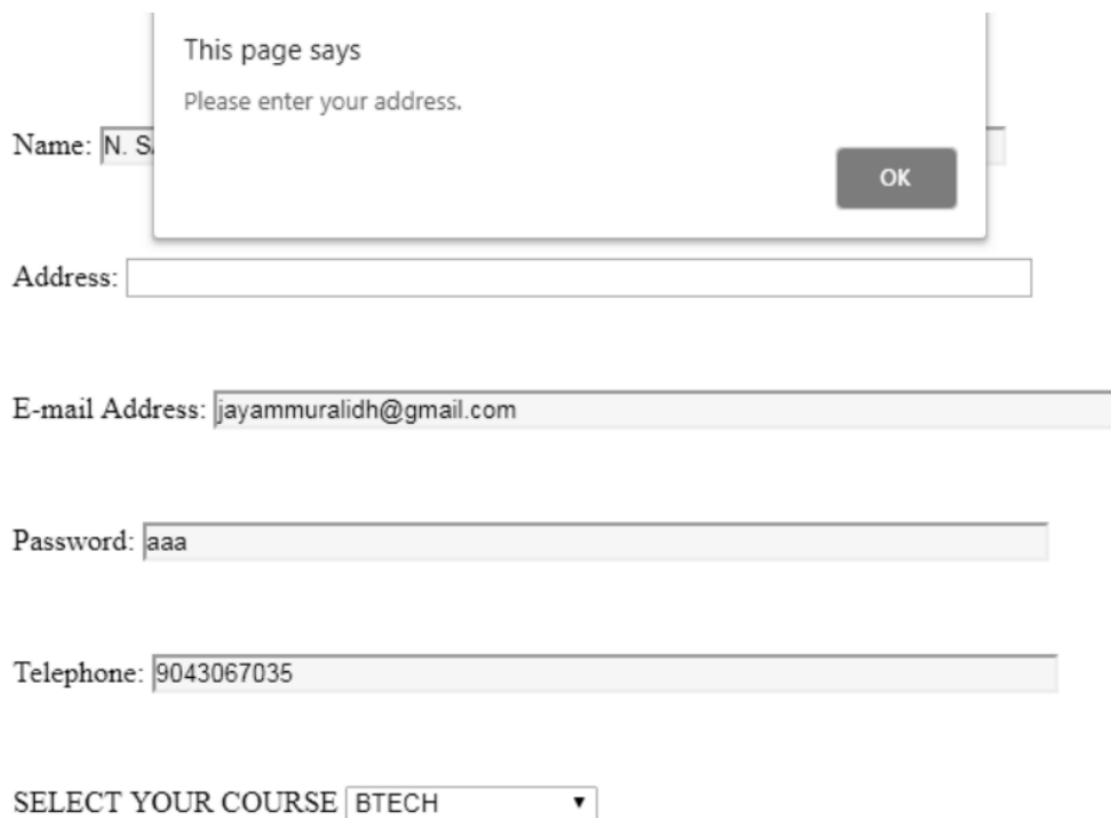
<body>
<h1 style="text-align: center"> REGISTRATION FORM </h1>
<form name="RegForm" action="submit.php" onsubmit="return VALIDATEDDETAIL()"
method="post">

    <p>Name: <input type="text" size=65 name="Name"> </p><br>
    <p> Address: <input type="text" size=65 name="Address"> </p><br>
    <p>E-mail Address: <input type="text" size=65 name="EMail"> </p><br>
    <p>Password: <input type="text" size=65 name="Password"> </p><br>
    <p>Telephone: <input type="text" size=65 name="Telephone"> </p><br>

    <p>SELECT YOUR COURSE
        <select type="text" value="" name="Subject">
            <option>BTECH</option>
            <option>BBA</option>
            <option>BCA</option>
            <option>B.COM</option>
            <option>VALIDATEDDETAIL</option>
        </select></p><br><br>
    <p>Comments: <textarea cols="55" name="Comment"> </textarea></p>
    <p><input type="submit" value="send" name="Submit">
        <input type="reset" value="Reset" name="Reset">

```

```
</p>  
</form>  
</body>  
</html>  
OUTPUT
```



The screenshot shows a web form with several input fields. A JavaScript alert box is displayed over the form, containing the text "This page says" and "Please enter your address." with an "OK" button. The form fields are as follows:

- Name: N. S
- Address: (empty text box)
- E-mail Address: jayammuralidh@gmail.com
- Password: aaa
- Telephone: 9043067035
- SELECT YOUR COURSE: BTECH (dropdown menu)

Result : The above program was executed successfully, and the output verified.

Ex. No:2**Fetch data and display the contents in the form of a card**

AIM: To Get data using Fetch API from an open-source endpoint and display the contents in the form of a card

Procedure:

1. Set up the HTML

- Create a container element in your HTML where the cards will be displayed. This could be a div with a specific ID or class.

2. Write the JavaScript Code

- Fetch the Data:
 - Use the fetch() method to make a request to the open-source API endpoint.
 - Handle the response:
 - Check for successful response status (e.g., response.ok).
 - Extract the data from the response using response.json().
- Create Card Elements:
 - Iterate through the received data array.
 - For each data item:
 - Create a new card element (e.g., a div with a class of "card").
 - Create HTML elements within the card to display the data (e.g., <h2> for titles, <p> for descriptions, for images).
 - Populate the elements with the corresponding data from the current item.
- Append Cards to the Container:
 - Append each created card element to the container element you defined in the HTML.

3. Style the Cards

- Use CSS to style the appearance of the cards (e.g., background color, borders, fonts, spacing).

script.js

```
// api url
const api_url =
  "https://employeeetails.sriventech.ac.in/my/api/path";

// Defining async function
async function getapi(url)
{

  // Storing response
  const response = await fetch(url);

  var data = await response.json();
  console.log(data);
  if (response)
  {
    hideloader();
  }
}
```

```

    show(data);
}
// Calling that async function
getapi(api_url);

// Function to hide the loader
function hideloader()
{
    document.getElementById('loading').style.display = 'none';
}

// Function to define innerHTML for HTML table
function show(data)
{
    let tab =
        `<tr>
          <th>Name</th>
          <th>Office</th>
          <th>Position</th>
          <th>Salary</th>
        </tr>`;
    // Loop to access all rows
    for (let r of data.list)
    {
        tab += `<tr>
          <td>${r.name} </td>
          <td>${r.office} </td>
          <td>${r.position} </td>
          <td>${r.salary} </td>
        </tr>`;
    }
    // Setting innerHTML as tab variable
    document.getElementById("employees").innerHTML = tab;
}

```

employee.html

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="script.js"></script>

    <meta charset="UTF-8" />
    <meta name="viewport"
      content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <!-- Here a loader is created which

```

```

    loads till response comes -->
<div class="d-flex justify-content-center">
  <div class="spinner-border"
    role="status" id="loading">
    <span class="sr-only">Loading...</span>
  </div>
</div>
<h1>Registered Employees</h1>
<!-- table for showing data -->
<table id="employees"></table>
</body>
</html>

```

OUTPUT

```

▼ Object
  list: Array(6)
    ▼ 0:
      name: "Billy Lee"
      office: "Detroit"
      position: "Web Developer"
      salary: "$50000"
      __proto__: Object
    ▼ 1:
      name: "John Doe"
      office: "Troy"
      position: "Manager"
      salary: "$90000"
      __proto__: Object
    ▼ 2:
      name: "James Baxter"
      office: "Detroit"
      position: "IT Support"
      salary: "$30000"
      __proto__: Object
    ▼ 3: {name: "Jimmy Lee", position: "Web Developer", office: "Detroit", ...}
    ▼ 4: {name: "Nick Wess", position: "Sales", office: "Ann Arbor", salar...}
    ▼ 5: {name: "Sarah Deets", position: "Graphic Designer", office: "Ann ...}
    length: 6

```

Result : The above program was executed successfully, and the output verified

Ex No 3

NodeJS server that serves static HTML and CSS files to the user without using Express.

Aim: To Create NodeJS server that serves static HTML and CSS files to the user without using Express.

Procedure

1. Import the server in a Node project
2. Place the following code in minimal-http-server/index.js
3. then import the server in your code
`const server = require('./minimal-http-server');`
4. Start the server as follows:
`server.init();`
5. The server will start on port 3000. Alternatively, you can specify the port as the first argument and hostname as the second argument to the `init()` method as shown below:
`server.init(4000, 'myserver');`

Program

```
/ Dependencies
const http = require('http');
const url = require('url');
const fs = require('fs');
const path = require('path');

// Container Object
const server = { };

// Base Directory - Assuming minimal-http-server will be accessed from its own folder.
const baseDir = path.join(__dirname, '../');

// Mime Types
const mimeTypes = {
  '.html': 'text/html',
  '.jpg': 'image/jpeg',
  '.css': 'text/css',
  '.js': 'text/javascript',
  '.png': 'image/png',
  '.ico': 'image/x-icon',
};

// Create a server
const httpServer = http.createServer((request, response) => {
  const parsedUrl = url.parse(request.url, true);

  // Remove leading and trailing / and \ from url path
  let pathName = parsedUrl.pathname;

  // Get the file extension
```

```

const extensionName = path.extname(pathName);
pathName = extensionName === undefined || extensionName === '' ?
`${pathName}/index.html` : pathName;
const responseContentType = getContentType(pathName);
response.setHeader('Content-Type', responseContentType);
// Read the file and send the response
fs.readFile(`${baseDir}${pathName}`, (error, data) => {
  if (!error) {
    response.writeHead(200);
    response.end(data);
  } else {
    console.log(error);
    response.writeHead(404);
    response.end('404 - File Not Found');
  } });});
// Get the content type for a given path
const getContentType = pathName => {
  // Set the default content type
  let contentType = 'application/octet-stream';

  // Set the contentType based on mime type
  for (var key in mimeTypes) {
    if (mimeTypes.hasOwnProperty(key)) {
      if (pathName.indexOf(key) > -1) {
        contentType = mimeTypes[key];
      } } }
  return contentType;
};

// Main method to be called to start the server. 'port' defaults to 3000 and 'host' defaults to
127.0.0.1
server.init = (port = 3000, host = '127.0.0.1') => {
  httpServer.listen(port, () => {
    console.log(`\x1b[32m%s\x1b[0m`, `Server is running at http://${host}:${port}`);
  });
};
module.exports = server;

```

OUTPUT

Bash

node server.js

This will start the server, and you should see the message Server listening on port 3000 in the console.

Result

The above program was executed successfully, and the output verified

Ex. No:4**Create a NodeJS server using Express that stores data from a form as a JSON file**

Aim : Create a NodeJS server using Express that stores data from a form as a JSON file and displays it in another page. The redirect page should be prepared using Handlebars.

Procedure :

1. Create a folder called fetch API in VS Code and create a new file JavaScript file. Now, write the Code to get free API and store it in Const api_url. Get the used id from Api using prompt method.
2. Convert into parseInt. Using async function get the API and point the needed data from the API to Card. New show function get the Emp details in the table format and send it to html file.
3. In HTML file create a Card using css file and Show the data fetched from the API. To the table. Using "Id - demo" link the css file, Script file to the html and open it in Live Server.

Server.js

```
const express = require('express');
const expbs = require('express-handlebars'); const app = express();
app.use(express.urlencoded());
app.engine('handlebars', expbs.engine ({defaultLayout: false,})); app.set('view engine',
    'handlebars');
//routing
app.get('/', function(request, response, next){ response.render('index', { layout: false });
    });
app.post('/', function(request, response, next){ response.send(request.body);});
app.listen(2000);
```

```
=>(main.handlebar)
```

```
<html>
  <head>
    <title>{{{ title }}}    </title>
  </head>
  <body>
    {{{ body }}}
  </body>
</html>
```

main.html

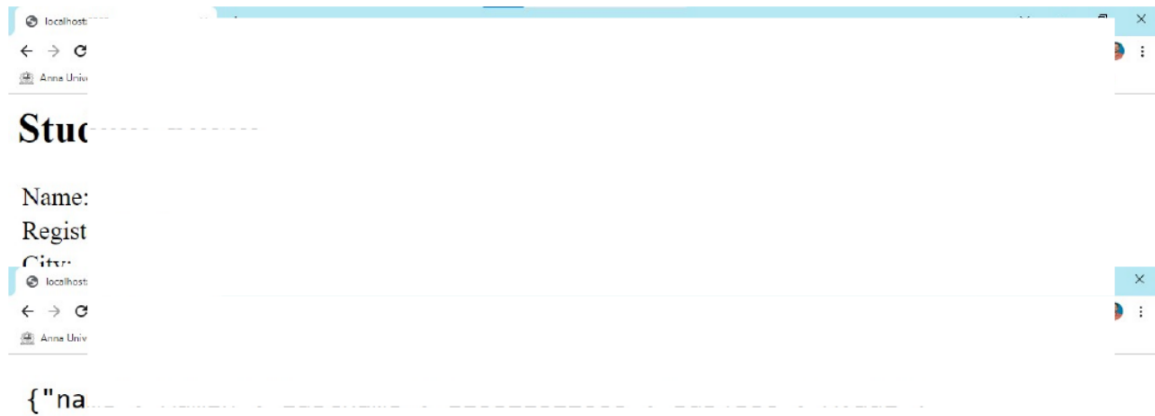
```
<html>
  <head>
    <title> {{{ title }}}    </title>
  </head>
  <body>
    {{{ body }}}
  </body>
</html>
```

index.html

```

<h1>Student Form</h1>
<form action="/" method="post">
  <Table style="font-size:20px;">
    <tr>
      <td><label for="name">Name:</label></td>
      <td><input type="text" id="fname" name="name" placeholder="Your
name.."></td>
    </tr>
    <tr>
      <td><label for="reg">Register Number:</label></td>
      <td><input type="text" id="lname" name="lastname"
placeholder="Your number"></td>
    </tr>
    <tr>
      <td><label for="city">City:</label></td>
      <td><input id="subject" name="subject" placeholder="Your City"
></input></td>
    </tr>
    <tr>
      <td><input type="submit" value="Submit"></td>
    </tr>
  </Table>
</form>

```

Output:

Result : The above program was executed successfully, and the output verified.

Ex. No:5**NodeJS Server creation with MongoDB**

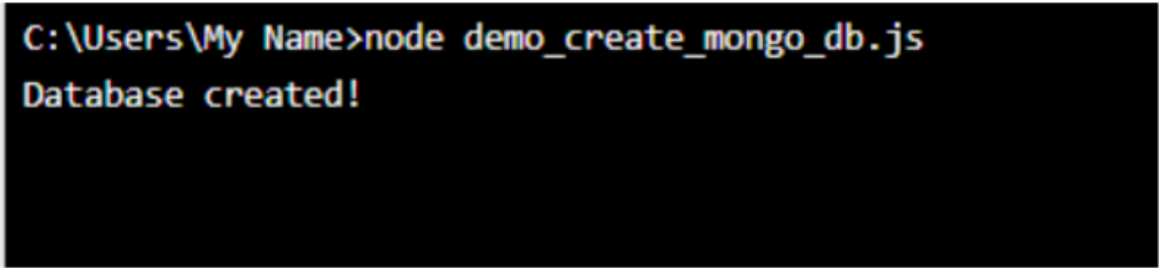
Aim: To Create a NodeJS server using Express that creates, insert, updates and deletes customers details and stores them in MongoDB database. The information about the user should be obtained from a HTML form.

Procedure:

1. Project Setup
 - Create a new Node.js project directory:
2. Create server.js file
3. Connect to MongoDB
4. Run the server

create_database.js

```
var MongoClient = require('mongodb').MongoClient;
//Create a database named "mydb":
var url = "mongodb://localhost:27017/mydb";
MongoClient.connect(url, function(err, db)
{
    if (err) throw err;
    console.log("Database created!");
    db.close();
});
```

Output


```
C:\Users\My Name>node demo_create_mongo_db.js
Database created!
```

Creating a Collection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    //Create a collection name "customers":
    dbo.createCollection("customers", function(err, res) {
        if (err) throw err;
        console.log("Collection created!");
        db.close();
    });
});
```

Output

```
C:\Users\My Name>node demo_mongodb_createcollection.js
Collection created!
```

Insert data into Collection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = { name: "Company Inc", address: "Highway 37" };
  dbo.collection("customers").insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
    db.close();
  });});
```

Output

```
C:\Users\My Name>node demo_mongodb_insert.js
1 document inserted
```

Update document

Update the document with the address "Valley 345" to name="Mickey" and address="Canyon 123":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: "Valley 345" };
  var newvalues = { $set: { name: "Michael", address: "Canyon 123" } };
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res) {
    if (err) throw err;
    console.log("1 document updated");
    db.close();
  });});
```

Output

```
C:\Users\My Name>node demo_update_one.js
1 document updated
```

Result : The above program was executed successfully, and the output verified.

Ex. No:6**NodeJS Server creation with MySQL**

Aim: To Create a NodeJS server that creates, reads, updates and deletes event details and stores them in a MySQL database. The information about the user should be obtained from a HTML form.

Procedure:

1. Create database in mysql
2. Write connection.php
3. Write php coding for reading data from database

```
<?php
include_once('connection.php');
$query="select * from emp where age>25";
$result=mysql_query($query);
?>
<!DOCTYPE html>
<html>
  <head>
    <title> Fetch Data From Database </title>
  </head>
  <body>

    <table align="center" border="1px" style="width:600px; line-height:40px;">
      <tr>
        <th colspan="4"><h2>Employee Record</h2></th>
      </tr>
      <t>
        <th>Empno</th>
        <th> Name </th>
        <th> Age </th>

      </t>
      <?php
        while($rows=mysql_fetch_assoc($result))
        {
          ?>
          <tr>
            <td><?php echo $rows['empno']; ?></td>
            <td><?php echo $rows['empname']; ?></td>
            <td><?php echo $rows['age']; ?></td>

          </tr>
        <?php
        }
      <?>
    </table>
  </body>
</html>
```

OUTPUT

+-----+-----+-----+-----+			
Empno	Empname	Age	
+-----+-----+-----+-----+			
1	John Poul	20	
2	Abdul S	25	
3	Sanjay	24	
+-----+-----+-----+-----+			

Result : The above program was executed successfully, and the output verified.

Ex. No:7**Create a counter using ReactJS**

Aim : To create a counter using ReactJS

Procedure :

1. Project Setup
 - Create a React App:
2. Component Creation
3. Render the Component
4. Run the Application

```

<!DOCTYPE html>
<html lang="en">

<body style="text-align:center">
  <h1>Fullstack</h1>
  <p>COUNTS</p>
  <div id="counter">
    <!-- counts -->
  </div>

  <script>
    let counts=setInterval(updated);
    let upto=0;
    function updated(){
      var count= document.getElementById("counter");
      count.innerHTML=++upto;
      if(upto===1000)
      {
        clearInterval(counts);
      }
    }
  </script>
</body>
</html>

```

OUTPUT

Fullstack

COUNTS

853

Result : The above program was executed successfully, and the output verified

Ex. No: 8 Create a Todo application using ReactJS

Aim: To Create a Todo application using ReactJS. Store the data to a JSON file using a simple NodeJS server and retrieve the information from the same during page reloads.

Procedure:

1. Set up the ReactJS frontend
2. Set up the Node.js backend
3. Create the todos.json file
4. Start the Node.js server
5. Connect Frontend and Backend
6. Test the App

Procedure:

1. Set up the ReactJS frontend
 1. Install Node.js and create a React application using the following command:


```
bash
npx create-react-app todo-app
cd todo-app
```
 2. Start the React development server:


```
bash
npm start
```
 3. Frontend Code: Create a simple interface that will allow adding, deleting, and showing to-do items.

Code:

App.js:

```
import React, { useState, useEffect } from 'react';
import './App.css';
```

```
function App() {
  const [todos, setTodos] = useState([]);
  const [newTodo, setNewTodo] = useState("");

  // Fetch the to-dos from the server when the component loads
  useEffect(() => {
    fetch('http://localhost:5000/api/todos')
      .then(response => response.json())
      .then(data => setTodos(data));
  }, []);

  const handleAddTodo = async () => {
    if (!newTodo) return;
    const newTodoItem = { id: Date.now(), text: newTodo };
    // Send the new to-do item to the server
    await fetch('http://localhost:5000/api/todos', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(newTodoItem)
    });
    setTodos([...todos, newTodoItem]);
    setNewTodo("");
  };
}
```

```

const handleDeleteTodo = async (id) => {
  await fetch(`http://localhost:5000/api/todos/${id}`, { method: 'DELETE' });
  setTodos(todos.filter(todo => todo.id !== id));
};

return (
  <div className="App">
    <h1>Todo App</h1>
    <input
      type="text"
      value={newTodo}
      onChange={(e) => setNewTodo(e.target.value)}
      placeholder="Enter a new todo"
    />
    <button onClick={handleAddTodo}>Add Todo</button>
    <ul>
      {todos.map(todo => (
        <li key={todo.id}>
          {todo.text}
          <button onClick={() => handleDeleteTodo(todo.id)}>Delete</button>
        </li>
      ))}
    </ul>
  </div>
);
}

```

export default App;

4. Style the App (Optional but nice for usability):
 - You can add CSS in App.css to style the to-do app.

2. Set up the Node.js backend

Now we need to create the backend server to handle the API calls and store/retrieve to-do data from a JSON file.

1. First, set up the server:

```

bash
mkdir backend
cd backend
npm init -y
npm install express body-parser fs

```

2. Server Code (server.js):

- Create a server.js file to manage API endpoints and interact with a todos.json file to store the to-do items.

```

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

```

```

const app = express();
const port = 5000;

```

```

// Middleware to parse JSON data
app.use(bodyParser.json());

```

```

    // Helper function to read todos from the JSON file
const readTodosFromFile = () => {
  try {
    const data = fs.readFileSync('todos.json', 'utf8');
    return JSON.parse(data);
  } catch (error) {
    return [];
  }
};

// Helper function to write todos to the JSON file
const writeTodosToFile = (todos) => {
  fs.writeFileSync('todos.json', JSON.stringify(todos, null, 2));
};

// API endpoint to get all todos
app.get('/api/todos', (req, res) => {
  const todos = readTodosFromFile();
  res.json(todos);
});

// API endpoint to add a new todo
app.post('/api/todos', (req, res) => {
  const todos = readTodosFromFile();
  const newTodo = req.body;
  todos.push(newTodo);
  writeTodosToFile(todos);
  res.status(201).json(newTodo);
});

// API endpoint to delete a todo by id
app.delete('/api/todos/:id', (req, res) => {
  const todos = readTodosFromFile();
  const updatedTodos = todos.filter(todo => todo.id !== parseInt(req.params.id));
  writeTodosToFile(updatedTodos);
  res.status(200).json({ message: 'Todo deleted' });
});

// Start the server
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});

```

3. Create the todos.json file:
 - Before running the server, create an empty todos.json file in the backend folder:

```
json
```
4. Start the Node.js server: Run the following command in the backend directory:

```
bash
node server.js
```

3. Connect Frontend and Backend

Ensure the frontend React app and backend Node.js server are running on different ports (React on port 3000, Node.js on port 5000). Your React app will be able to make requests to the backend API for storing and retrieving todos.

4. Test the App

1. Open the React app (<http://localhost:3000/>).
2. Add, delete, and refresh the page.
 - The To-Dos will persist across page reloads because they are stored in todos.json on the Node.js server.

Output

```
+-----+
|      Todo App      |
+-----+
| [Enter a new todo here] [Add Todo] |
+-----+
| - Buy Groceries [Delete]   |
| - Walk the Dog  [Delete]   |
+-----+
```

Ex. No: 9. Create a Sign up and Login mechanism**Aim:**

Create a simple Sign up and Login mechanism and authenticate the user using cookies. The user information can be stored in either MongoDB or MySQL and the server should be built using NodeJS and Express Framework.

Procedure:

1. **Set up Project Directory** Create a new directory for your project.
2. **Initialize Project** Initialize the project with npm and install necessary dependencies.
3. **Create Basic File Structure**
4. **Connect to MongoDB** Update server.js to set up Express server and MongoDB connection.
5. **Create User Model** In models/User.js, define the schema for users.
6. **Create Auth Routes** In routes/auth.js, create routes for signing up and logging in.
7. **Start Server** Run the server
8. **Testing the Application**

Program:**Steps to Create the Sign-up and Login Mechanism**

1. **Set up Project Directory** Create a new directory for your project.
 bash
 mkdir auth-app
 cd auth-app
2. **Initialize Project** Initialize the project with npm and install necessary dependencies.
 bash
 npm init -y
 npm install express mongoose bcryptjs cookie-parser jsonwebtoken
 - express: Web framework for Node.js
 - mongoose: MongoDB object modeling
 - bcryptjs: Library for hashing passwords
 - cookie-parser: Middleware to handle cookies
 - jsonwebtoken: To create JWT tokens for authentication
3. **Create Basic File Structure**
 Create the following files:
 - server.js: Entry point for the app
 - models/User.js: MongoDB schema for user
 - routes/auth.js: Routes for sign-up and login
4. **Connect to MongoDB** Update server.js to set up Express server and MongoDB connection.
 javascript

```
const express = require("express");
const mongoose = require("mongoose");
const cookieParser = require("cookie-parser");
const app = express();
const PORT = 5000;

// Middleware
app.use(express.json());
app.use(cookieParser());

// Connect to MongoDB
mongoose.connect("mongodb://localhost:27017/auth-app", {
```

```

    useUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log("Connected to MongoDB"))
  .catch((err) => console.log("Failed to connect to MongoDB", err));

```

// Routes

```

const authRoutes = require("./routes/auth");
app.use("/api/auth", authRoutes);

```

// Start Server

```

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

```

5. **Create User Model** In models/User.js, define the schema for users.

```

const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");

```

```

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

```

// Hash password before saving user

```

userSchema.pre("save", async function(next) {
  if (!this.isModified("password")) {
    return next();
  }
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

```

```

userSchema.methods.comparePassword = async function(password) {
  return bcrypt.compare(password, this.password);
};

```

```

const User = mongoose.model("User", userSchema);
module.exports = User;

```

6. **Create Auth Routes** In routes/auth.js, create routes for signing up and logging in.
javascript

```

const express = require("express");
const jwt = require("jsonwebtoken");
const User = require("../models/User");

```

```

const router = express.Router();

```

// Sign-up route

```

router.post("/signup", async (req, res) => {
  const { username, password } = req.body;

```

```

try {
  // Check if user already exists
  const existingUser = await User.findOne({ username });
  if (existingUser) {
    return res.status(400).json({ message: "User already exists" });
  }

  // Create new user
  const newUser = new User({ username, password });
  await newUser.save();

  res.status(201).json({ message: "User created successfully" });
} catch (error) {
  res.status(500).json({ message: "Error creating user" });
}
});

// Login route
router.post("/login", async (req, res) => {
  const { username, password } = req.body;

  try {
    const user = await User.findOne({ username });
    if (!user) {
      return res.status(400).json({ message: "Invalid credentials" });
    }

    const isMatch = await user.comparePassword(password);
    if (!isMatch) {
      return res.status(400).json({ message: "Invalid credentials" });
    }

    // Create JWT token
    const token = jwt.sign({ userId: user._id }, "secretKey", {
      expiresIn: "1h",
    });

    // Set cookie
    res.cookie("authToken", token, {
      httpOnly: true,
      secure: process.env.NODE_ENV === "production", // Only for https
    });

    res.json({ message: "Login successful" });
  } catch (error) {
    res.status(500).json({ message: "Error logging in" });
  }
});

module.exports = router;

```

7. **Start Server** Run the server with:
bash

node server.js

Testing the Application

To test the application, use a tool like Postman or Insomnia.

1. Sign Up

- Make a POST request to `http://localhost:5000/api/auth/signup` with the following JSON body:

json

```
{
  "username": "testUser",
  "password": "password123"
}
```

2. Login

- Make a POST request to `http://localhost:5000/api/auth/login` with the following JSON body:

json

```
{
  "username": "testUser",
  "password": "password123"
}
```

- You should receive a successful login response with a cookie named `authToken`.

Output

```
{
  "message": "User created successfully"
}
```

Login Request: After logging in, you should receive:

```
{
  "message": "Login successful"
}
```


Ex. No: 10**Create and Deploy a virtual machine**

Aim : To Create and Deploy a virtual machine using a virtual box that can be accessed from the host computer using SSH

Procedure:

1. Prepare your computer for virtualization. Install Hypervisor (virtualization tool).
2. Import a virtual machine.
3. Start the virtual machine.
4. Use the virtual machine.
5. Shutdown the virtual machine

VIRTUALIZATION – the underlying technology that allows a virtual operating system to be run as an application on your computer's operating system.

HYPERVISOR – the virtualization application (such as VirtualBox or VMware) running on your host computer that allows it to run a guest virtual operating system.

HOST – the computer on which you are running the hypervisor application.

GUEST – a virtual operating system running within the hypervisor on your host computer. The virtual operating system term is synonymous with other terms such as Virtual \ Machine, VM and instance.

Program:

Step 1: Prepare your computer for Virtualization:

- **Enable Processor Virtualization:** Ensure Virtualization is enabled on your computer. See the Virtualization Error (VT-d/VT-x or AMD-V) for troubleshooting support.
- **Review File Sync Services** for tools like OneDrive, Nextcloud, DropBox Sync, iCloud, etc. If you are using a data synchronization service, make sure it DOES NOT (or at least not frequently) synchronize the folder in which your hypervisor imports and installs the Virtual Machines.
- **File sync services** can cause a dramatic fall-off in performance for your entire system as these services try to synchronize these massive files that are getting updated constantly while you are using the Virtual Machines.
- **Sufficient Disk Space:** Virtual Machines require a significant amount of Disk space (10 GB or more each is typical). Ensure you have sufficient space on your computer.
- **Admin Privileges:** Installing a hypervisor on a host in most cases requires admin privileges.

Step 2: Install Hypervisor (Virtualization Tool):

Installing a hypervisor on your host is usually quite simple. In most cases, the install program will ask only a couple of questions, such as where to install the hypervisor software.

Step 3: Import a Virtual Machine:

- The first step is to download the Virtual Machine for your course from our Course Virtual Machines page. This will download an .ova file. The .ova file is actually a compressed (zipped) tarball of a Virtual Machine exported from Virtual Box.
- Once the Virtual Machine has been imported, it will normally show up in the guest list within your hypervisor tool.

Step 4: Start the Virtual Machine:

To start up a Virtual Machine guest in most hypervisors, you simply click on the desired guest and click the Start button (often double-clicking the guest icon will work as well).

Step 5: Using the Virtual Machine:

- Sharing files between the guest and host: To learn about different ways of sharing files, check out this guide.
- Run a command with sudo (root) privileges: Open a terminal and type any command with sudo in front to run that command as root.
- Example: `sudo apt-get install vim` – will install the vim text editor package on an Ubuntu Linux Virtual Machine.
- Find the IP address of your guest: Open a terminal and type `ifconfig | more` – The `|` more (pronounced “pipe more”) will “pipe” the output of the `ifconfig` command to the `more` command, which will show the results one page at a time, so it doesn’t scroll by before you see it all.
- If you have a Host-Only Network IP address, you will see an IP of 192.168.56.101 (or something similar). Check the Trouble-Shooting section below for more information about the Host-Only Network.

Step 6: Shut down the Virtual Machine:

When you are done using a guest Virtual Machine, regardless of hypervisor, you need to shut it down properly. This can be done in three ways:

1. Press the shutdown button found on the desktop, taskbar, or task menu of the guest operating system.
2. Open a terminal and type the command: `sudo shutdown -h now`
3. In the guest window, click Machine (menu) -> ACPI Shut down – This will simulate the power button being pressed

Output:

Bash

`ssh [USERNAME]@[IP_ADDRESS]`

Enter Password: Enter the password for the specified user.

SSH Connection: A screenshot of the terminal window on the host computer, displaying a successful SSH connection to the guest OS.

RESULT:

The above program is executed successfully. Hence output verified

Ex. No. 11. Create a docker container that will deploy a NodeJS ping server using the NodeJS image.

Aim

To create a Docker container that deploys a simple Node.js ping server using the Node.js Docker image

Procedure

1. Install Docker
2. Create a Directory for the Project
 - Create a new directory where you'll store your project files.

```
bash
```

```
mkdir nodejs-ping-server
```

```
cd nodejs-ping-server
```

3. Create the Node.js Ping Server Code

- Inside the directory, create a file called server.js with the following code to create a simple "ping" server.

```
server.js:
```

```
javascript
```

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

```
const hostname = '0.0.0.0';
```

```
const port = 8080;
```

```
const server = http.createServer((req, res) => {
  if (req.method === 'GET' && req.url === '/ping') {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify({ message: 'pong' }));
  } else {
    res.statusCode = 404;
    res.end('Not Found');
  }
});
```

```
server.listen(port, hostname, () => {
```

```
  console.log(`Server running at http://${hostname}:${port}/`);
```

```
});
```

This will create a server that listens on port 8080, and if you make a GET request to /ping, it will return { "message": "pong" }.

4. Create a Dockerfile

- Now create a file named Dockerfile in the same directory to define the Docker image.

5. Build the Docker Image

- After creating the Dockerfile, build the Docker image using the following command:

```
bash
```

```
docker build -t nodejs-ping-server .
```

6. Run the Docker Container

- Once the image is built, you can run the Docker container:

```
bash
```

```
docker run -d -p 8080:8080 --name ping-server nodejs-ping-server
```

7. Test the Server

- To test if the server is working, use curl or a browser to access the /ping route:

Program Code

1. server.js

```
javascript
```

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

```
const hostname = '0.0.0.0';
```

```
const port = 8080;
```

```
const server = http.createServer((req, res) => {
  if (req.method === 'GET' && req.url === '/ping') {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify({ message: 'pong' }));
  } else {
    res.statusCode = 404;
    res.end('Not Found');
  }
});
```

```
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

2. Dockerfile

Dockerfile

```
# Use official Node.js image from Docker Hub
```

```
FROM node:16
```

```
# Set the working directory inside the container
```

```
WORKDIR /usr/src/app
```

```
# Copy the package.json and package-lock.json if available
```

```
# (skip this part if you don't need dependencies for this example)
```

```
# COPY package*.json ./
```

```
# Install dependencies (skip if no dependencies)
```

```
# RUN npm install
```

```
# Copy the server.js file to the working directory
```

```
COPY server.js .
```

```
# Expose the port the app will run on
```

```
EXPOSE 8080
```

```
# Run the Node.js server
```

```
CMD ["node", "server.js"]
```

Output

After running the server, you should be able to access the endpoint via `http://localhost:8080/ping`, and it should respond with the following JSON message:

```
json
```

```
{"message": "pong"}
```

```
$ curl http://localhost:8080/ping
```

```
{"message": "pong"}
```

FSWD Lab Viva Voce Questions

➤ General JavaScript and Web Development

1. What is the purpose of form validation, and why is it important in JavaScript?
2. Can you explain how JavaScript is used to validate the contents of a form?
3. What are some common validation techniques used in form validation using JavaScript?
4. How does the Fetch API work, and what are its advantages over XMLHttpRequest?
5. How do you handle errors while using the Fetch API in JavaScript?
6. What is the difference between synchronous and asynchronous requests, and how does Fetch work asynchronously?
7. What is the purpose of creating cards to display content dynamically. How would you structure such content in HTML and CSS?
8. What are some performance considerations when retrieving and displaying data from an API?
9. How would you prevent a form from being submitted if the validation fails in JavaScript?
10. What are JSON objects, and how do they play a role in API interactions?

➤ Node.js Basics

11. How can a Node.js server serve static HTML and CSS files without using Express?
12. What modules are required to set up a basic HTTP server in Node.js?
13. What is the role of the fs module in serving static files in Node.js?
14. How would you handle routing and serving HTML files in a basic Node.js server?
15. Can you explain the role of the http module in Node.js and how it can be used to create a simple web server?
16. What are the benefits of using Express over using only the built-in HTTP module in Node.js?
17. How does the req (request) and res (response) objects work in a Node.js HTTP server?

➤ Form Handling and Data Storage

18. How would you store form data as a JSON file on the server in a Node.js application?
19. What is the purpose of using Handlebars, and how does it help with rendering dynamic content in Express?
20. Can you describe how to handle form submissions in Express and redirect the user to a different page?
21. How does Express handle POST requests to submit form data?

22. How you would store form data in a JSON file and retrieve it later in an Express-based Node.js application.
23. What is the purpose of `res.redirect` in Express, and how does it work with routing?
24. Can you explain how to use Handlebars for displaying dynamic content after form submission in Express?

➤ **CRUD Operations with Databases**

25. How would you set up MongoDB for use with a Node.js application?
26. Can you explain how to perform CRUD (Create, Read, Update, Delete) operations in a MongoDB database with Node.js?
27. How do you structure MongoDB data, and what kind of data format is typically used?
28. What are the differences between SQL and NoSQL databases, and why would you choose one over the other?
29. How do you set up a MongoDB database connection in a Node.js Express app?
30. What is Mongoose, and how is it used in Node.js to interact with MongoDB?
31. How would you handle error handling and validation in MongoDB CRUD operations in Node.js?
32. Can you explain how to use the `find()` method in MongoDB with Mongoose to fetch records?
33. How do you update existing records in MongoDB using Mongoose?
34. Can you explain how to delete a document in MongoDB using Mongoose in Node.js?

➤ **MySQL and Event Management**

35. How do you set up a MySQL database in a Node.js application?
36. What are the steps to perform CRUD operations with a MySQL database using Node.js?
37. Can you explain how to connect a Node.js application to a MySQL database using the `mysql` or `mysql2` package?
38. How do you structure the data for an event management system in MySQL?
39. What is the role of prepared statements in MySQL when working with Node.js?
40. How would you retrieve event details from a MySQL database and display them on an HTML page?

➤ **React.js Basics**

41. What is the concept of state in React, and how does it differ from props?
42. How would you implement a counter application using React? What are the key concepts involved?

- 43. How can you update state in React and trigger re-rendering of a component?
- 44. What are controlled and uncontrolled components in React, and how do they differ?
- 45. How does React handle events, and how can you bind event handlers to components?
- 46. Can you explain how React handles reactivity in the user interface (UI)?
- 47. What are hooks in React? Explain the use of the useState and useEffect hooks.
- 48. How do you use the useEffect hook to manage side effects in a React application?

➤ **React.js and Node.js Integration**

- 49. How would you implement a simple Todo application with React where data is stored in a JSON file using a Node.js server?
- 50. What strategies would you use to persist the Todo list data on page reloads?

TBS: Servlet application to display the examination result of a student

AIM:

To write a java program to display the academic performance of a student using servlet application.

PROCEDURE:

Step-1: Create the “Myservlet” folder in “c:\program files\Apache Software

Foundation\Tomcat6.0\Webapps” location. Now create another folder

“WEB-INF” inside “MyServlet” folder. Create one more folder

“classes” inside “WEB-INF” folder

Step-2: Type the servlet program and save it in your login(Z:\drive)

Step-3: Compile the servlet program

Step-4: Copy the class file and paste it in the “c:\program files\Apache Software

Foundation\Tomcat 6.0\web apps\MyServlet\WEB-INF\classes folder

Step-5: Open notepad and type the xml file and save it in a file “web.xml” under

“c:\program files\apache software Foundation\Tomcat 6.0\webapps\MyServlet\

“WEB-INF” folder

Step-6: Create a html file and save it in the z:\drive

Step-7: Open the html file in the browser. Enter the Roll No of a student

And click the “Best of Luck!!!” button and view the result

PROGRAM:

Step – 1: Create the “MyServlet” folder in “C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps” location. Now create another folder “WEB-INF” inside “MyServlet” folder. Create one more folder “classes” inside “WEB-INF” folder.

Step – 2: Type the following Servlet program and save it in your login area (Z:\ drive).

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class StudDet extends HttpServlet
{
    static int i;
    Connection con;
    PrintWriter out;
    ResultSet rs;
    public void init()
    {
```

```

        i=0;
        con=null;
        out=null;
        rs=null;
    }
    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
        ServletException,IOException
    {
        i++;
        response.setContentType("text/html");
        out=response.getWriter();
        out.println("<b>You are user no. "+ i +" to visit this site.</b><br><br>");
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con = DriverManager.getConnection
            ("jdbc:oracle:thin:@172.16.58.1:1521:dbserver", "scott", "tiger");
            PreparedStatement ps=null;
            String query = "select rollno,name,dept,m1,m2,m3 from student where
                rollno=?";
            ps = con.prepareStatement(query);
            ps.setInt(1,Integer.parseInt(request.getParameter("RegNo")));
            rs=ps.executeQuery();
            out.println("<b><center>Student Details</center></b><br><br>");
            ResultSetMetaData rsmd=rs.getMetaData();
            int colcount=rsmd.getColumnCount();
            out.println("<table border=1>");
            out.println("<tr>");
            for(int i=1;i<=colcount;i++)
            out.println("<th>"+rsmd.getColumnLabel(i)+"</th>");
            out.println("</tr>");
            if(rs.next())
            {
                out.println("<tr>");
                out.println("<td>"+rs.getInt("rollno")+"</td>");
                out.println("<td>"+rs.getString("name")+"</td>");
                out.println("<td>"+rs.getString("dept")+"</td>");
                out.println("<td>"+rs.getInt("m1")+"</td>");
                out.println("<td>"+rs.getInt("m2")+"</td>");
                out.println("<td>"+rs.getInt("m3")+"</td>");
                out.println("</tr>");
            }
            out.println("</table>");
            out.println("</body>");
        }
        catch(Exception e)
        {

```

```

        out.println(e.toString());
    }
}
public void destroy()
{
    try
    {
        i=0;
        rs.close();
        con.close();
        out.close();
    }
    catch(Exception e)
    {
        out.println(e.toString());
    }
}
}

```

Step – 3: Compile the Servlet program

Step – 4: Copy the class file and paste it in the “C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\MyServlet\WEB-INF\classes” folder.

Step – 5: Open Notepad, type the following and save it in a file “web.xml” under “C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\MyServlet\WEB-INF” folder.

```

<?xml version="1.0"?>
<web-app>
  <servlet>
    <servlet-name>StudentDetails</servlet-name>
    <servlet-class>StudDet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>StudentDetails</servlet-name>
    <url-pattern>/StudDet</url-pattern>
  </servlet-mapping>
</web-app>

```

Step – 6: Create a HTML file in the Z:\ drive with the following.

```

<html>
<head>
  <title>
    Find Student Information
  </title>
</head>
<body>
  <form method="GET" action="http://localhost:8080/MyServlet/StudDet">
    <h2 align=center>Find Student Information<center></h2>

```

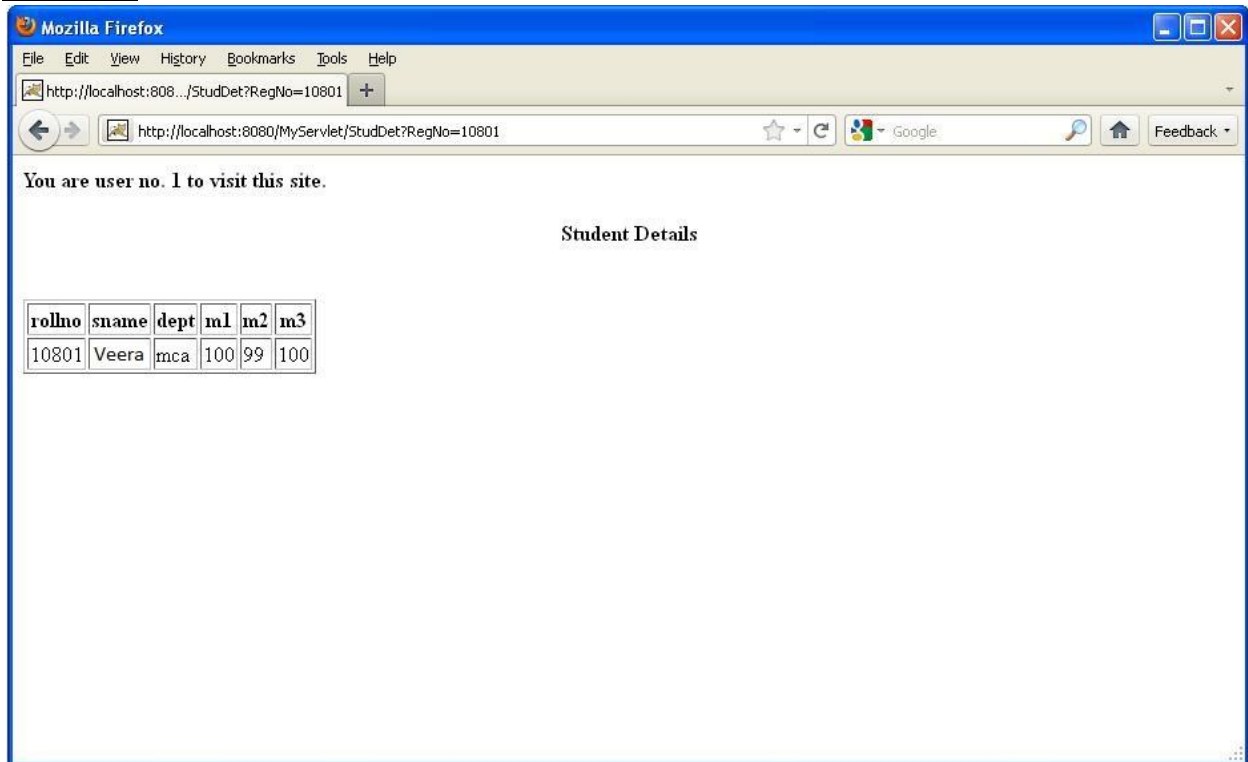
```

Enter Student Roll No.
<input type="text" name="RegNo">
<input type="submit" value="Best of Luck!!!">
</form>
</body>
</html>

```

Step – 7: Open the HTML file in a browser, enter the Roll No. of a student, click the “Best of Luck!!!” button and view the result.

OUTPUT:



RESULT:

Thus the above program was executed and output verified.